République Algérienne Démocratique et Populaire

Ministère de l'enseignement supérieur et de la recherche scientifique

Université d'El-Oued

Faculté des sciences exactes

Département d'informatique

Support de Cours

Module : Bases de données

Canevas de formation de deuxième année licence informatique (S4)

Spécialité : Systèmes Informatiques (SI)

Par
Saci Medileh

Année universitaire: 2023/2024

Module :
Bases de données

Deuxième année licence Informatique

Spécialité: Système informatique SI

الملخص:

هذا الدرس يهدف إلى تحقيق عدة أهداف أساسية، تتعلق بفهم واكتساب المهارات في مجال قواعد البيانات. يشمل ذلك فهم أهداف قواعد البيانات، الهندسة اللازمة لإنشائها، واللغات المستخدمة للتفاعل معها. يتعلم الطالب أيضًا كيفية التحكم في هياكل البيانات والتعامل معها بفعالية.

وتشمل المهارات التي يهدف الدرس إلى تحقيقها معرفة الطالب بالمفاهيم الأساسية لقواعد البيانات والتحليل المناسب لاحتياجات المستخدمين. يتم توجيه الدرس أيضًا لتعليم الطالب كيفية تصميم قواعد البيانات، بدءًا من تحديد الهياكل والعلاقات الأساسية وصولًا إلى تنفيذ استعلامات متقدمة وإدارة عمليات المعاملات.

يعتمد المحتوى التعليمي لهذا المقياس على النموذج العلاقي لقواعد البيانات ويعتمد بشكل خاص على استخدام لغة SQL (Structured Query Language) التي تعد لغة معيارية وشائعة في تفاعل المستخدمين مع قواعد البيانات. يتم تسليط الضوء على المفاهيم النظرية والخوارزميات الأساسية التي تحقق التفاعل الفعّال والمستدام مع البيانات، بدءًا من تصميم البيانات وصولاً إلى التعامل مع الاستعلامات وإدارة عمليات المعاملات.

Résumé

Ce cours vise à atteindre plusieurs objectifs fondamentaux liés à la compréhension et à l'acquisition de compétences dans le domaine des bases de données. Cela inclut la compréhension des objectifs des bases de données, l'ingénierie nécessaire à leur création, ainsi que les langages utilisés pour interagir avec elles. Les étudiants apprendront également à contrôler les structures de données et à les manipuler efficacement.
Les compétences visées par le cours incluent la connaissance des concepts de base des bases de données et une analyse appropriée des besoins des utilisateurs. Le cours vise également à enseigner aux étudiants comment concevoir des bases de données, en commençant par définir les structures et les relations fondamentales, jusqu'à la mise en œuvre d'interrogations avancées et la gestion des opérations transactionnelles.

Le contenu pédagogique de ce module s'appuie sur le modèle relationnel des bases de données et met particulièrement l'accent sur l'utilisation du langage SQL (Structured Query Language), une langue standard et largement utilisée dans l'interaction des utilisateurs avec les bases de données. Les concepts théoriques et les algorithmes de base permettant une interaction efficace et durable avec les données sont mis en avant, couvrant la conception des données, le traitement des requêtes et la gestion des transactions.

الكلمات المفتاحية:

عربيّـة: تحليل الأنظمة الكلاسيكية؛ قواعد البيانات؛ النموذج العلائقي؛ الارتباط الوظيفي؛ الاشكال المعيارية; لغة الاستعلام SQL.

فرنسيـة: Approche classique; Bases de données ; SGBD; Dépendance fonctionnelle ; Modèle relationnel ;Langage SQL. ; Les formes normales;

Spécialité : Systèmes Informatiques
Parcours type : Licence informatique – Académique
Semestre : S4
Libellé de l'U.E. : UEF2.2.2, 10 Crédits, Coefficient : 06
Module : Bases de données, 05 Crédits, Coefficient : 03
VHG : (Cours :1h30, TD :1h30, TP :1h30) × 15 semaines
Chargé de cours et TD : Mr. MEDILEH Saci

Fiche de Cours : Bases de données

Objectif :

Comprendre les objectifs, les architectures et les langages de bases de données. Maîtriser les fondements théoriques et les algorithmes de base des systèmes de gestion de bases de données, depuis la conception de base de données jusqu'au traitement de requêtes et la gestion de transactions. Le module s'appuie sur le modèle relationnel et les langages associés, en particulier SQL.

Evaluation :

- Contrôle continu (40%) :
  - Travaux dirigés : Préparation des travaux, participation dans les discussions en classe, compréhension des concepts traités, présence. (30%)
  - Interrogation écrites : Des questions sur une partie de cours, au minimum deux interros, durée de 20 à 30 min. (40%)
  - Travaux pratiques: Se familiariser avec un SGBD relationnel. (30%)

- Examen final (60%): Examen portant sur l'ensemble de cours et travaux.

Programme :

Chapitre 1 : Généralités sur les bases de données
- L'approche classique d'informatisation
- L'approche base de données
- Le système de gestion de bases de données.

Chapitre 2 : Le modèle relationnel
- Objectifs visés par le modèle relationnel
- Démarche de conception d'un schéma relationnel
- Dépendances fonctionnelles
- Formes normales d'une relation
- Méthodes de conception d'un schéma

Chapitre 3 : l'algèbre relationnelle
- Langages de manipulation de données relationnelles
- Les opérations de bases
- Les opérations complémentaires
- Propriétés des opérations algébriques

Chapitre 4 : Présentation générale de SQL
- Les requêtes en SQL
- Utilisation des sous requêtes
- Les fonctions de groupes

Références :

- Brahim Belattar, Cours de bases de données (2009), Université de Batna, Algérie.
- Georges Gardarin, Bases de données, Edition Eyrolles, 2003
- Philippe Rigaux, Cours de bases de données (juin 2001), CNAM, Paris.
  J.Akoka et I.Comyn-Wattiau (2001), Conception des bases de données relationnelles en pratique, Vuibert, Paris.
- Serge Abiteboul, Foundations of databases, ADDISON WESLEY Publishing Company Incorporated, 1995.
- Solange Ghernaouti-Hélie et Yves Pigneur, Notions et principes généraux d'informatique: Base de données II, (Support de cours), Université de Lausanne.
- Christine CAMPIONI, Cours de bases de données, Aix-Marseille Université.
- Les meilleurs cours, tutoriels et Docs sur les SGBD et le SQL, Site développez.com.
- Le Web…

❶ Système d'information et informatique

- **Le système d'information d'une organisation (au sens large: société, entreprise, institution, club, groupe structuré ...) regroupe tout ce qui à quelque niveau que se soit traite ou stocke des informations relatives à l'organisation concernée.**

- **Le système d'information comprend des informations relatives:**

  - aux flux : produits en stocks, produits commandés, bons de livraison, factures, bons de commandes...
  - à l'univers extérieur: clients, fournisseurs...
  - à l'organisation de l'entreprise: que se passe-t-il entre l'enregistrement d'une commande et sa livraison?
  - aux contraintes légales: lois, règlements, paramètres financiers...
  - etc.

- **Dans l'approche d'un système d'information, on distingue :**
  - la formalisation des données (informations de toute nature) présentes à un moment ou un autre dans le système,
  - la formalisation des traitements (i.e des processus dynamiques) intervenant dans le système.
- **La formalisation des données mène à l'élaboration d'un modèle de données .**
- **La formalisation des traitements se traduit par la définition de modèles de traitements .**
- **Ce travail de formalisation d'un système d'information constitue ce que l'on appelle : une analyse informatique.**

- **De manière générale seule une partie du SI peut être automatisée. Cette partie sera appelée Système Automatisé d'Information (SAI)**
- **Qu'est-ce qui peut être informatisé?**
  - **Mémorisation d'informations:**
    - » stockage des données,
    - » structuration des données,
    - » ...
  - **Traitements:**
    - » contrôle des données,
    - » mise à jour,
    - » recherches,
    - » calculs.
  - **Interface entre l'univers et le SI: saisie, accès.**
- **Le SAI communique avec son environnement extérieur par des saisies et des accès. Il contient une partie mémorisation et une partie traitements.**

Pourquoi l'analyse informatique?

- La difficulté essentielle dans la réalisation du SAI réside dans le fait qu'elle **concerne un nombre important de personnes**, de caractéristiques très variées (la direction, le service informatique, les responsables de service, les utilisateurs terminaux) d'où la nécessité **d'une méthode**.

- **Les premières informatisations se sont souvent révélées catastrophiques (la liste suivante n'est pas exhaustive):**

  - logiciel ne fonctionnant pas ou ne réalisant pas ce pourquoi il avait été prévu,
  - logiciel incapable d'évoluer avec le matériel, la taille de l'entreprise, les changements dans l'entreprise,
  - études correctes sur le papier mais impossibles à implanter,
  - logiciel correct mais mettant trop de temps à délivrer les résultats,
  - informatisation bouleversant l'organisation humaine de l'entreprise, rejetée par le personnel...

L'approche classique de mise en place d'une application informatique

➢ **Dans une entreprise** L'approche **classique** de mise en place d'une **application informatique** consistait le plus souvent à :

- ✓ l'**écriture** d'un certain nombre de **programmes**
- ✓ **programmes** destinés à **l'exploitation** d'un ensemble **de fichiers** qu'il fallait **aussi créer**.

➢ **Les principaux problèmes** posés par cette démarche sont :

- **la redondance** des informations
- **la dépendance entre les données et les programmes qui les manipulent.**

La Redondance des informations

⇨ **Redondance** = une même information peut se trouver dans **plusieurs** fichiers distincts.

Ex : l'identité d'un employé (nom, prénom, adresse, etc.)

- ➢ peut figurer dans un fichier **F1** propre à une application **P1** calculant la paie,
- ➢ et dans un fichier **F2** propre à une autre application **P2** de gestion des remboursements des frais médicaux.

⌦ **Cause :**

- création **anarchique** de fichiers
- **absence de coordination** entre les groupes de développement **entraîne une duplication non contrôlée** des informations

Conséquences :

La redondance **complique** les opérations de mise à jour

- il est nécessaire pour **une même information mise à jour dans un fichier** de la **mettre à jour dans tous les fichiers** où elle est supposée exister.

Ex :

- si on **change l'adresse** d'un employé dans le **fichier F1**,
- **il faut faire aussi ce changement** dans tous les fichiers contenant **l'adresse de cet employé** .

- Ceci devient **de plus en plus difficile** dès que le **nombre de fichiers** est **élevé**
- Souvent on a **tendance à retarder** **la mise à jour** dans les autres **fichiers** ce qui engendre une situation **incohérente**

situation incohérente :

- l'interrogation de la même information par deux **utilisateurs différents** **chacun** sur **son propre fichier** fournira **2 résultats différents.**

La Dépendance entre les données et les programmes

⌦ Généralement, la **création d'un fichier** sous entend la réalisation d'un **certain nombre d'opérations** telles que :

- La structuration des enregistrements :

  définir les champs, leurs types et leur ordre

- Le choix d'une organisation :

  séquentielle, séquentielle indexée, directe, etc.

- Le choix du support :

  disque dur, bande magnétique, etc.

☝ Cette **création** était donc **figée** car faite en fonction d'un ou de plusieurs programmes.

- Les **données contenues** dans les **fichiers** sont **directement associées aux programmes** qui les exploitent
- par le biais d'une description contenue dans ces programmes eux-mêmes.

Exemple :

- Pour mettre en place une application de gestion de prêts dans laquelle on suppose qu'un **emprunteur peut emprunter jusqu'à trois (3) ouvrages**, on doit créer un fichier dont les enregistrements auront à peu près la structure suivante :

Nom	Prénom	Adresse	Ouvrage 1	Ouvrage 2	Ouvrage 3

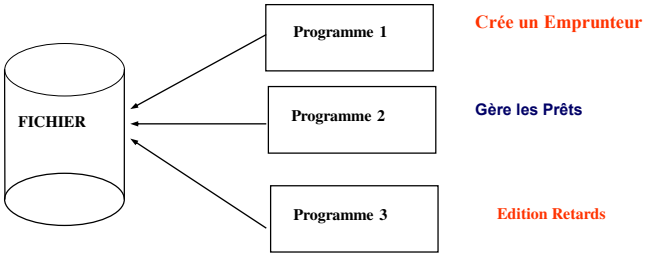
- Les programmes qui utilisent un tel fichier en donnent la description à l'intérieur du programme même.

Exemple :

Si on se réfère par exemple au langage COBOL, une telle description aurait l'allure suivante :

```
01      PRET
        02      NOM             PICTURE         A(20)
        02      PRENOM          PICTURE         A(20)
        02      OUVRAGE         OCCURS   3      TIMES
                03              COTE_OUVRAGE    9(6)
```


- Si **ce fichier** est utilisé par les programmes P1, P2, P3



- **Si on doit le modifier pour le besoin de P2** : prise en compte de l'âge de l'emprunteur par exemple,
  - ⇨ il **faudra modifier P1 et P3** en conséquence.

- Ceci est dû au fait que tous les programmes **voient** le fichier de la même façon **bien qu'ils n'aient pas tous besoin des mêmes informations.**

⌦ un **changement de l'organisation** du fichier sur disque pour le besoin d'un programme **Pi** (souci d'optimisation des temps d'accès de **Pi**)

- **obligera les autres programmes à prendre en compte ce changement même s'ils n'ont pas vraiment besoin de ce type d'organisation.**

➢ une **réorganisation des champs de l'enregistrement** (changement de l'ordre des champs à l'intérieur de l'enregistrement) entraînera une modification dans les programmes et une reconstruction du fichier sur le support.

⌦Ceci montre qu'il **existe une dépendance étroite** entre les données et les programmes qui les manipulent

➢ **Elle se traduit par le fait suivant :**

il n'est **pas possible** de changer la structure et **l'organisation** des données **sans modifier** en conséquence les **programmes**.

⌦il **manque** un **moyen de filtrer les informations** afin de ne **retenir** que **celles** qui sont **utiles** à **un programme donné**.

L'approche Bases de données

Notions générales

Définition

Une **BD** un **ensemble d'informations structurées non redondant** permettant la **mise en place d'une série d'applications** informatiques **destinées** à une **grande variété d'utilisateurs**.

- **Exemple:**

Dans une entreprise, les informations concernant son fonctionnement :

- Employés
- Produits Fabriqués
- Moyens matériels (Machines, Véhicules, Magasins, etc.)

peuvent être rassemblées sous forme de BD et mises à la disposition de nombreux utilisateurs (cadres de l'entreprise, gestionnaires, opérateurs, clients, fournisseurs, etc.).

Objectifs d'une base de données

Parmi les principaux objectifs visés par une base de données :

➢ **Permettre le Partage de l'information**

Une **BD** permet le **partage** d'un **ensemble unique d'informations** par plusieurs utilisateurs.

☝ **il faut que cette mise en commun soit faite :**

1. tout en préservant **la vue particulière** que chaque utilisateur peut avoir des informations,
2. et en s'assurant que la **simultanéité des traitements** qui peuvent être effectués ne risque pas **d'altérer l'intégrité** de la base de données.

➢ **Organiser les données indépendamment des programmes :**

⌧ Afin de construire un ensemble d'informations structurées non redondant et qui soit partageable par plusieurs utilisateurs :

- il est nécessaire de **faire abstraction des traitements** particuliers de tel ou tel utilisateur (ou programme)
- **organiser les informations en fonction** de leur **nature** et des **liens réels** qui existent entre elles et non en fonction des traitements.

➔ C'est de cette manière qu'on arrivera à garantir le maximum d'indépendance entre données et programmes.

Avantages de l'approche B.D.

- ✓ Contrairement aux approches classiques, **l'approche BD est le reflet d'une évolution dans la gestion** de l'entreprise.
- ✓ **Elle rend possible :**

La centralisation de l'information :

l'information n'est plus éparpillée dans différents fichiers à différents endroits

L'intégration :

tout ce qui se fait dans un service est visible par d'autres services

La diffusion de l'information archivée :

si l'information est disponible à un seul endroit, **elle est facile à diffuser**

Ceci a pour avantages :

1- Améliorer la cohérence de l'information :

une seule valeur pour une même information

2- Réduire les redondances :

une même information n'est stockée si possible qu'une seule fois

3- Réduire les efforts de saisie et de mise à jour :

conséquence de 2 . une information qui doit être stockée une seule fois ne sera saisie qu'une seule fois. De même que sa mise à jour ne se fera qu'une seule fois

Démarche de conception d'une BD

- la conception d'une base de données doit se faire en utilisant une méthode de conception qui définit la démarche à suivre.
- Plusieurs méthodes de conception existent

Exemple :

la méthode **MERISE** basée sur le modèle **E/R** de Chen

- Pour certaines méthodes, on dispose même d'un outil logiciel d'aide à la conception appelé aussi un Atelier de Génie Logiciel (AGL)
  - un ensemble de logiciels permettant l'automatisation d'un certain nombre de tâches lors des différentes phases du processus de conception :
    - génération automatique de la structure de la B.D.,
    - de programmes d'accès et de manipulation,
    - etc..

- Quelle que soit la méthode utilisée, la conception d'une base de données passe par **un processus de modélisation** :
  - Modéliser une certaine **partie du monde réel** afin de caractériser les entités qu'on manipule (étudiants, Comptes Bancaires, Ouvrages, etc.)
  - Caractériser les **attributs de ces entités** en fonctions des problèmes que doit résoudre l'existence de la B.D. :
    - Gestion de la scolarité,
    - Gestion de Prêt d'ouvrages dans une bibliothèque,
    - etc.

Exemple :

- Pour un étudiant on aura besoin d'un attribut relatif au résultat dans un module dans le cas d'une **Gestion de la scolarité,** alors que ce type d'attribut ne sera pas nécessaire ? dans le cas d'une application de **Gestion de Prêt**

➢ Le cas le plus général est celui où la B.D. est partagée par plusieurs utilisateurs.

- Ces utilisateurs n'ont pas tous la même vue des données de la base,
- Ils n'ont pas tous **à voir la base dans sa totalité**
- Chaque utilisateur n'est concerné que par une partie de celle-ci.

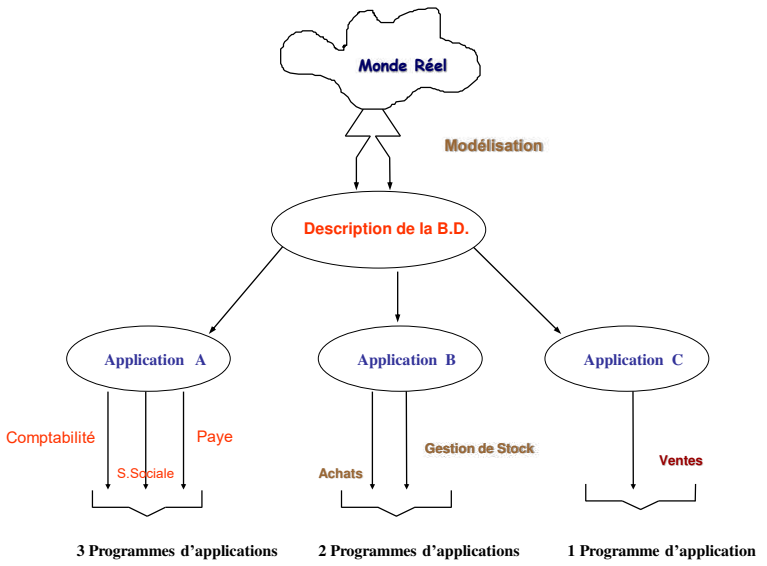
Exemple :

Dans une entreprise l'ensemble des informations sur les :

- départements,
- les employés,
- les produits,
- le matériel,
- etc.

- peuvent être rassemblées sous forme d'une B.D. et il est bien rare qu'un utilisateur de cette base ait besoin de toutes ces informations à la fois.

Processus de Modélisation d'une B.D.



Remarque

☝ Il serait plus juste de remplacer le terme **Utilisateur** par celui d'**Application**

➔ On définit en général une B.D. afin de mettre en place une série d'applications ayant chacune ses programmes et ses propres utilisateurs exploitant le même sous-ensemble de données.

Exemple :

Dans une B.D. de Gestion Universitaire, on peut mettre en place une application de gestion de Prêts qui sera **composée d'un ensemble de programmes** et dont les utilisateurs seront ceux concernés par les traitements réalisés par ces programmes (Secrétaire, Documentaliste, Archiviste, etc.).

Problèmes posés par la Centralisation de l'information sous forme de B.D.

- la **centralisation** de l'information sous forme **d'une B.D. unique** pose un certain nombre de problèmes liés directement à l'**intégrité** et à la **sécurité** de ces informations.
- **Parmi ces problèmes**, on peut énumérer les suivants :

❖ Nature et type de l'information

➢ Lors de la mise en place d'une B.D., **l'utilisateur décrit les propriétés que doivent vérifier ses données** et qui **doivent être préservées tout au long de l'existence de la base de données.**

❑ **Ces propriétés peuvent se situer à différents niveaux :**

- **Appartenance d'une donnée à un ensemble de valeurs**

  Ex : l'âge d'un employé est un entier positif compris entre 0 et 150
  0< âge < 150

- **Déclaration de propriétés invariantes au cours du temps**

  Ex : Un enseignant à une heure Donnée ne peut se trouver que dans une seule salle

- **Relation d'ordre total à respecter lors du stockage des données**

  Ex : Les employés doivent être stockés dans la B.D. par ordre croissant de leur numéro ou par ordre alphabétique sur leur nom

❖ Sûreté physique, sûreté de fonctionnement et point de reprise

➢ Il s'agit de protéger l'information contre **un mauvais fonctionnement** soit de la **machine**, soit du **système** (logiciel) qui géré la base de données.

❑ Dans le **premier cas**, on peut **délimiter les enregistrements** qui ont été altérés ou perturbés

❑ Dans le **second cas** le problème est beaucoup **plus complexe**.

– Une des solutions en usage consiste à **prendre à intervalles réguliers des copies de la B.D.** et à enregistrer l'ensemble des transactions (opérations) effectuées sur la base.

– Ceci permettra en cas d'incident de **régénérer une copie** consistante (i.e. sans défauts) de la base.

❖ Partage de l'information

➢ Lorsque **deux programmes** P1 et P2 **veulent se partager** la même donnée A, il peut y avoir **perte d'intégrité**.

Exemple :

P1 accède à	*A*	et la transfère dans son buffer propre
P2 accède à	*A*	et la transfère dans son buffer propre
P1 modifie	*A*	dans son buffer puis la recopie dans la B.D.
P2 modifie	*A*	dans son buffer puis la recopie dans la B.D.
		venant ainsi écraser les modifications faites par P1.

➢ **La solution** à ce problème serait par exemple celle de l'exclusion mutuelle qui est une technique utilisée dans les systèmes d'exploitation.

❖ Problème des données confidentielles

- Il s'agit de protéger les données contre des **utilisateurs indiscrets**
- On dispose en général pour cela de **procédures sélectionnant les accès à la base de données.**

➢ Lorsqu'un utilisateur veut faire un accès, on distingue **deux phases** :

1. **La phase d'identification :**

   a pour but **d'identifier l'utilisateur qui veut se connecter** à la base de données. Ceci est possible grâce à un mot de passe, une carte spéciale, etc.

2. **La phase d'autorisation :**

   après identification de l'utilisateur, **permet de déterminer ce que peut faire cet utilisateur sur telle ou telle données** (consulter seulement, consulter et mettre à jour, etc.)

Niveaux de description d'une Base de Données

- La description d'une base de données peut se faire à **différents niveaux**, suivant que l'on regarde plus du côté de l'utilisateur que du côté du stockage des données sur les supports physiques.

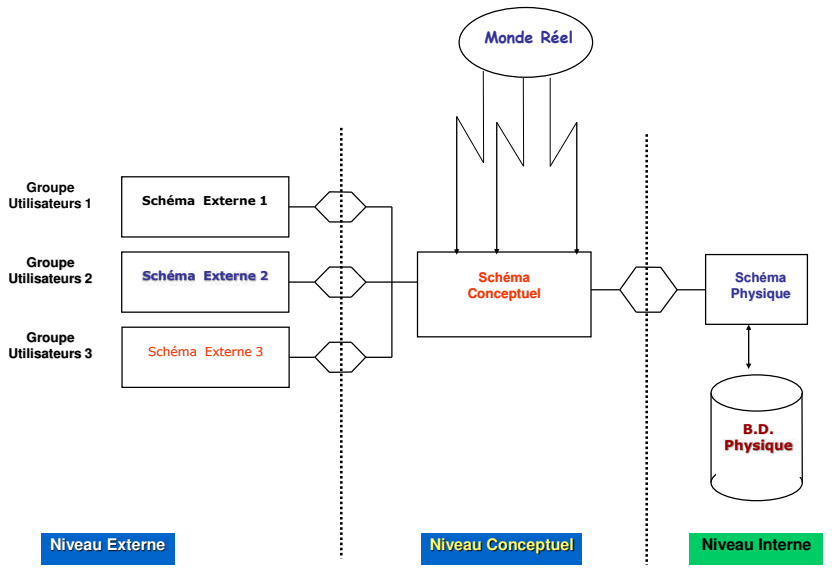
- **On distingue communément trois (3) niveaux de description d'une B.D.**

1- niveau conceptuel

2- niveau externe

3- niveau interne

Niveaux de description d'une Base de Données



Le Niveau conceptuel

- Le **schéma conceptuel** est la **partie fondamentale** dans l'architecture d'une base de données.
- Il a pour but de **décrire en termes abstraits mais fidèles** une certaine réalité d'une organisation et de ses processus de gestion qui ont nécessité la mise en place d'une B.D.
- **L**e passage du **monde réel au schéma conceptuel** correspond à un **processus de modélisation** :

  les objets du monde réel ayant les mêmes caractéristiques sont classés en catégories et désignés par des noms (Etudiants, Véhicules, etc..
- **D**ans le processus de modélisation, le concepteur de la B.D. spécifie le schéma conceptuel en utilisant les possibilités offertes par un **modèle de données.**

Définition d'un modèle de données

- Un modèle de données est un **outil formel** destiné à **décrire la réalité** de manière **indépendante** de **tout traitement informatique**.
- Un modèle de données doit permettre de regrouper les objets du monde réel auxquels on s'intéresse en classes d'objets de nature identique.

Exemple :

- Dans une application de gestion universitaire, on pourra regrouper les étudiants dans une classe d'objets qu'on appellera ETUDIANTS et les modules dans une autre classe d'objets qu'on appellera MODULES.
- Par la suite on ne fera référence aux objets de ces classes que par l'intermédiaire de ces noms.

- Un modèle de données doit aussi permettre de décrire les **liaisons** ou **associations** qui peuvent exister entre les classes d'objets.

Exemple :

- Dans une application de gestion universitaire, l'inscription est un phénomène qui associe un objet étudiant appartenant à la classe ETUDIANTS à un ou plusieurs objets modules appartenant à la classe MODULES.
- On pourra dans ce cas créer une association qu'on nommera **INSCRIPTION** entre la classe ETUDIANTS et la classe MODULES afin de modéliser cette réalité.

➢ **Un schéma conceptuel** est donc le **résultat d'un processus de modélisation** fait **en respectant** les possibilités d'un **modèle de données.**

Classification des modèles

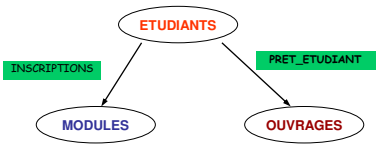
➢ Il existe **trois grandes classes de modèles** de données qui se **distinguent par** la nature des associations qu'ils permettent de modéliser.

Ce sont :

- ⊠ Les modèles hiérarchiques
- ⊠ Les modèles réseaux
- ⊠ Les modèles **relationnels**

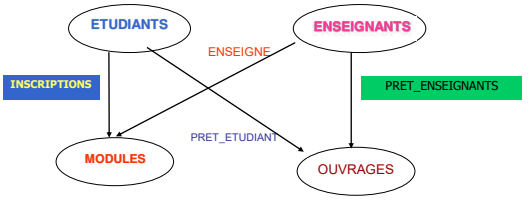
Le modèle hiérarchique

- A l'aide du modèle hiérarchique, le schéma conceptuel peut être vu comme **un graphe arborescent** dont **les nœuds** correspondent aux classes d'objets (**entités**) et les **arcs** entre deux nœuds aux **liaisons** ou associations entre les entités.
- Un tel graphe possède donc **un nœud racine** (sur lequel n'arrive aucun arc !) et les autres nœuds sont des fils, petit-fils, etc., de cette racine.
- Avec le modèle hiérarchique, le nombre de flèches pouvant arriver sur un nœud est donc égal à un (sauf pour le nœud racine).



Le modèle réseau

- A l'aide de ce modèle, le schéma conceptuel peut être vu comme un graphe général où les nœuds correspondent aux classes d'objets et les arcs entre deux nœuds aux associations.
- A la différence du modèle hiérarchique on peut avoir ici **plusieurs arcs** qui **arrivent sur le même nœud**.
- De même que la notion de **nœud racine n'existe pas** avec le modèle réseau.



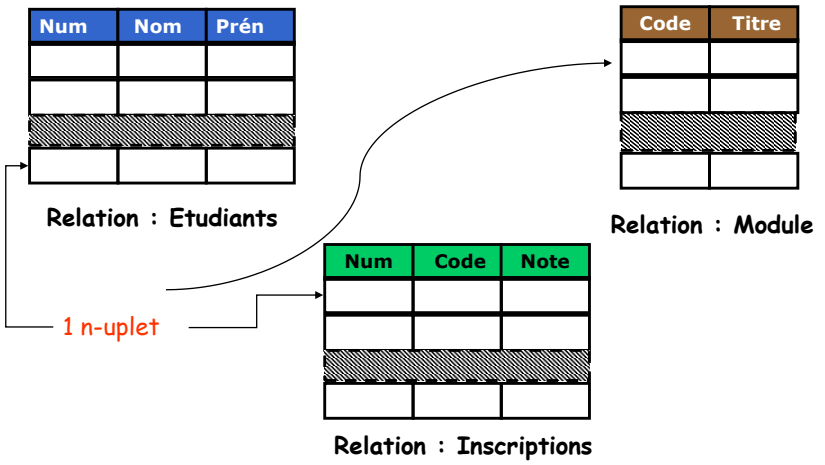
Le modèle relationnel

- Ce modèle est fondé sur la théorie mathématiques des relations.
- Le **schéma conceptuel** peut être vu comme **un ensemble de tables** (ou relations) à **n colonnes**, **n** désignant **le degré de la relation.**
- Avec le modèle relationnel, **une table** sert à représenter **aussi bien** une **classes d'objets** qu'une **association** entre des classes d'objets.

Ainsi, la distinction entre nœud et arc comme dans les autres modèles n'est pas nécessaire avec le modèle relationnel.

- Chaque élément d'une table est appelé un **n-uplet**.

Par exemple la table **INSCRIPTIONS** décrit l'association entre la classe d'objets **ETUDIANTS** et la classe **MODULES** et qui permet de modéliser le fait qu'un étudiant peut s'inscrire à 0, 1 ou plusieurs modules.



Le Niveau Externe

- Ce niveau correspond à la vision de tout ou partie du schéma conceptuel par un groupe d'utilisateurs concerné par une application.

- **I**l s'agit de décrire à l'aide d'un **schéma externe ou Vue** la façon dont seront perçues les données par un programme d'application.

Exemple :

Dans une base de données de gestion universitaire, un groupe d'utilisateurs concerné par les INSCRIPTIONS des étudiants

- n'a pas besoin d'avoir une vision globale de la base et peut se limiter à la partie qui englobe les informations relatives aux **étudiants et aux Modules.**

Le Niveau Externe

- Un schéma externe peut donc être considéré comme un sous schéma du schéma conceptuel.

- Grâce à cette notion de schéma externe, **chaque groupe d'utilisateurs perçoit les données à sa façon**

  - Par exemple, une donnée vue comme donnée numérique par un groupe peut être vue comme une chaîne de caractères par un autre (cas d'une date par exemple).

  - Un groupe peut ne pas voir certaines caractéristiques (attributs) d'une entité (ex : note obtenue dans un module) qui seront par contre visibles par un autre groupe (ou application).

Le Niveau Interne

- Ce niveau a pour but de spécifier **comment les données** sont **stockées** sur les **supports physiques**.
- Cette spécification est faite par le biais d'un **schéma physique** ou **schéma de stockage**.
- Ce schéma permettra par exemple de :
  - **Décrire la structure des fichiers qui constituent la base de données** (nom d'un fichier, organisation, adresse sur le support, etc.)
  - **Définir les méthodes d'implantation** (fichier plat, inversé, etc.)
  - **Préciser les chemins d'accès aux enregistrements** (index, chaînage, calcul d'adresse, etc.)

- Il faut noter que tous ces aspects ne doivent pas affecter les applications **sauf sur le plan des performances** d'accès à la base
- Afin de garantir l'un des objectifs visés par une base de données à savoir : **l'indépendance entre les données et les programmes**.
- Le souci du schéma physique est donc de **pouvoir changer l'organisation physique des données** sans modifier le schéma conceptuel ni les programmes d'application.
- Par exemple, pour augmenter les performances d'accès à la base de données, on peut être amené à :
  - Changer l'organisation d'un fichier (Passer par exemple d'une organisation initialement séquentielle à une organisation séquentielle indexée ou directe).
  - Déplacer physiquement le fichier vers une autre adresse sur le support
  - Modifier les chemins d'accès aux enregistrements (changer d'index, ajouter d'autres indexes, etc.).

❶ Fondements du Modèle Relationnel

Caractéristiques du Modèle

- Le modèle relationnel est un modèle de données qui consiste à percevoir la base de données comme un ensemble de relations qu'on peut visualiser sous forme de tables à deux dimensions :
  - les colonnes qui correspondent aux attributs d'une relation
  - et les lignes qui correspondent aux tuples
- La caractéristique principale du modèle : utilise qu'un seul concept : **la relation.**

Caractéristiques du Modèle

- Contrairement aux modèles réseau et hiérarchique qui distinguent entre les concepts d'entité et de lien,

⇒ Avec le modèle relationnel : on modélise indifféremment une entité ou un lien entre deux entités par une relation .

- De plus, les associations de type **N : M** sont directement supportées par le modèle relationnel sans aucune transformation préalable **comme c'est le cas avec les modèles réseau et hiérarchique.**

⇒ **L'intérêt de cette approche est qu'elle conduit à un modèle simple, plus facile à comprendre et à utiliser même par un utilisateur non spécialiste.**

Définitions

Attribut

Un attribut désigne une propriété ou une caractéristique d'une relation

Domaine d'un attribut

- **Le domaine d'un attribut correspond à l'ensemble des valeurs que peut prendre cet attribut.**
- **Cet ensemble peut être fini ou dénombrable comme il peut être infini.**
- **En pratique, il est très difficile d'énumérer l'ensemble des valeurs définissant le domaine d'un attribut surtout si cet ensemble est infini.**
- **La majorité des SGBD assimilent le type de donnée d'un attribut (entier, date , chaîne de caractères, réel,...) à son domaine et limitent ainsi la définition d'un attribut à la déclaration du nom de l'attribut suivi de son type :**

(ex : Prénom_Employé : Char(10)**).**

Produit cartésien d'un ensemble de domaines

- **Le produit cartésien d'un ensemble de domaines D1 , D2 , D3 ,....., Dn non nécessairement distincts**

  que l'on note : D1 x D2 x D3 x.....x Dn

  est l'ensemble des tuples (v1 , v2 , v3 ,....., vn)

  tel que vi ∈ Di et ce pour tout i = 1, 2, 3,....., n.

Produit cartésien d'un ensemble de domaines

Exemple : Soit les trois domaines suivants :

D1= (Lundi, Mardi, Mercredi, Jeudi, Vendredi, Samedi, Dimanche)

D2= (1, 2 , 3 ,.....31)

D3= (Janvier, Février, Mars, Avril, Mai, ...Décembre)

Alors le produit cartésien D1 x D2 x D3=

{ (Lundi, 1, Janvier), (Lundi, 2, Janvier),… (Lundi, 31, Janvier),

(Lundi, 1, Février), ... (Lundi, 31, Février), ...

(Lundi, 1, Décembre)……..(Lundi,31, Décembre),

(Mardi, 1, Janvier), ... , (Mardi,31, Décembre)……

(Dimanche, 1, Janvier), ... (Dimanche, 31, Décembre) }

Relation

Définition formelle

- **Une relation est définie par une liste d'attributs A1 , A2 , A3 ,...., An ayant respectivement pour domaine D1 , D2 , D3 ,..... , Dn .**
- **On la note R(A1 , A2 , A3 ,..... , An) ou R est le nom de la relation.**
- **Elle est composée d'un ensemble de tuples (a1 , a2 , a3 ,....., an) ou ai ∈ Di ∀ i = 1, 2, 3,....., n,**

  cet ensemble constitue un sous-ensemble du **produit cartésien** : D1 x D2 x D3 x..... x Dn.

Relation

Tuple d'une relation

- lorsqu'on parle de **relation** on utilise les termes de **tuple** (ou n-uplet) et d'**attribut**.
- Lorsqu'on parle de **table** c'est à dire la représentation sous forme tabulaire d'une relation, on utilise les termes de **ligne** et de **colonne**.

  - Un tuple d'une relation désigne tout simplement une ligne dans la table représentant la relation.
  - Un attribut quant à lui désigne une colonne dans la même table.

Relation

Arité d'une relation

- L'arité d'une relation est le nombre de ses attributs.
- Par exemple, la relation **R(A1 , A2 , A3 ,..... An)** a une arité égale à **n** car elle possède **n** attributs.

Comme cas particuliers il faut distinguer :

- les relations ayant un seul attribut (tables à une seule colonne) et qui sont dites unaires (i.e. arité = 1)
- et les relations ayant deux attributs seulement (tables à deux colonnes) et qui sont dites binaires (i.e. arité = 2).

Relation

Cardinalité d'une relation

- La cardinalité d'une relation est le nombre de tuples de cette relation.
- La cardinalité d'une relation **R** sera donc un nombre entier et se note : |**R**|.

Relation

Schéma d'une relation

- Le schéma d'une relation est le nom de la relation suivi de la liste des attributs de cette relation

  Exemple : **R(A1 , A2 , A3 ,..... An)**.

- C'est ce qu'on appelle aussi **la définition en intention** de la relation

Relation

Extension d'une relation

- L'extension d'une relation est l'ensemble des tuples de la relation
- C'est ce qu'on appelle aussi **la définition en extension** d'une relation
- Le terme **extension** fait donc référence au **contenu** de la relation
- alors que le terme **intention** fait référence au **contenant** c'est à dire le **schéma de la relation**

Exemple

La relation :

CATALOGUE_PRIX (Code_Produit , Désignation , Fournisseur , Prix).

est définie par ses attributs qui sont :

- **Code_Produit ,**
- **Désignation ,**
- **Fournisseur**
- **et Prix.**

Son nom est : CATALOGUE_PRIX.

Une extension de cette relation :

Code_Produit	Désignation	Fournisseur	Prix
P001	ETAGERE	ALI	1500
P002	ARMOIRE	ALI	5400
P003	TIROIR	ALI	750
P030	FAUTEUIL	BRAHIM	3000
P003	TIROIR	KAMEL	600
P150	CHAISE	KAMEL	1000
P150	CHAISE	OMAR	900

Ce tuple indique par exemple que le produit dont le code est P002 et la désignation est ARMOIRE peut être fourni par le fournisseur qui s'appelle ALI qui le vend au prix de 5400.

L'extension de la relation permet de faire les remarques suivantes :

- **Il n'y a pas deux tuples identiques**
  - **2 tuples sont identiques si les valeurs de tous leurs attributs respectifs sont égales.**
- **L'ordre d'apparition des tuples n'a pas de signification particulière (i.e. ordre des lignes)**
  - **si ceci n'empêche pas que les tuples soient classés selon un critère appliqué aux valeurs d'un attribut (ordre alphabétique des noms de fournisseurs, ordre croissant des codes de produits, etc.).**
  - **Par exemple les tuples de la relation CATALOGUE_PRIX** sont classés selon un ordre alphabétique de l'attribut FOURNISSEUR désignant le nom d'un fournisseur.

- **L'ordre des attributs n'a pas de signification particulière. (i.e. ordre des colonnes)**

  ⇒ **les noms donnés aux attributs de la relation doivent être distincts**

❖ Soit la relation COMPOSITION (Pièce, Pièce, Quantité) qui modélise le fait qu'une pièce p **est composée** d'une certaine quantité q d'une autre pièce p'.

Cette relation est défini sur trois attributs dont les deux premiers possèdent le même nom Pièce signifiant implicitement qu'ils ont le même domaine.

- **Problème Posé :**

  – il sera difficile pour ne pas dire impossible d'associer un sens à ces deux attributs en fonction de leur ordre d'apparition

    c'est à dire le premier attribut de la relation signifie la pièce composée, le deuxième attribut de la relation signifie la pièce composante ou inversement.

- **Le seul moyen pour distinguer entre les attributs sera donc le nom qui leur sera donné.**

  - Dans ce cas, il faudra donner aux attributs des noms différents selon le rôle que joue chaque attribut dans la relation.

Exemple,

- **COMPOSITION** (Pièce_Composée, Pièce_Composante, Quantité).

Clé d'une relation

Informellement :

- La clé d'une relation R est un sous-ensemble d'attributs **X** dont les valeurs identifient un tuple et un seul de la relation R.
- **A tout moment la clé d'une relation possède les propriétés suivantes :**

  – Unicité : elle identifie un seul tuple de la relation

  – Composition minimale : aucun attribut de la clé ne peut être éliminé sans détruire la propriété d'unicité.

Clé d'une relation

Remarque :

- D'après la définition d'une relation, celle-ci ne peut pas contenir deux tuples identiques (deux tuples différent au moins par les valeurs d'un attribut).

⇒ Dans le cas où on est pas en mesure d'identifier une clé pour une relation donnée, **il faut rappeler que toute relation** possède au moins une clé à savoir : **l'ensemble de ses attributs.**

•

Clé candidate

- Une relation peut posséder plusieurs clés (des sous-ensemble d'attributs) satisfaisant toutes la définition donnée plus haut.

⇒ Chacune de ces clés sera appelée **une clé candidate**.

Notion de clé primaire

- Lorsqu'on dispose pour une relation donnée de plusieurs clés candidates, il est nécessaire de ne **retenir qu'une seule** parmi cet ensemble

La clé retenue s'appellera alors la **clé primaire**.

- Elle sera utilisée effectivement pour repérer de manière unique les tuples de la relation.
- La clef primaire devra impérativement être déclarée au moment de la description ou de la définition de la relation au niveau du SGBD.

Notion de clé primaire

- **Le choix de la clé primaire** est généralement effectué en fonction des deux critères suivants :
  - On choisit la clé candidate ayant le plus petit nombre d'attributs : **il est préférable de minimiser le nombre d'attributs composant la clé.**
  - On choisit la clé candidate en fonction de son usage pour la localisation des tuples : **il s'agit de privilégier la clé candidate dont l'usage serait le plus fréquent pour localiser les tuples de la relation.**

Notion d'attribut primaire

- On appelle **attribut primaire** tout attribut **appartenant à une clef candidate** (ou à plusieurs clés en même temps) d'une relation.

- **Par** exemple si la relation **R(A, B, C, D) possède 2** clefs candidates (A,B) et (A,C) ,
  - **les attributs primaires sont : A, B et C.**
  - l'attribut D n'appartient à aucune clé candidate**. Un tel attribut sera dit** attribut non primaire**.**
  - L'attribut primaire A **appartient en même temps aux deux clés candidates de R.**

Remarques

- Toute clé candidate qui n'a pas été retenue **comme clé primaire** constitue ce que l'on appelle une **clé secondaire**.
- Sur le plan pratique, la clé primaire et les clés secondaires peuvent **servir à définir des index** qui permettent de réaliser des accès sélectifs aux tuples de la relation.
- il faut souligner que cette distinction entre clé primaire et clé secondaire ou candidate n'a d'intérêt que sur le plan théorique :

  Au niveau du SGBD on peut en général construire un index sur tout attribut ou ensemble d'attributs même s'il ne constitue pas une clé.

Démarche de conception d'un schéma relationnel

- Si on ne dispose pas d'une aide au niveau de la modélisation, ni d'une métrique pour juger de la qualité du schéma conceptuel,
- il n'est pas exclu que des erreurs de conception aient lieu et peuvent conduire à un schéma mal conçu qui **peut poser des problèmes** lors de l'exploitation de la base de données.

⇒ L'un des objectifs principaux visés par le modèle relationnel :

est de mettre à la disposition du concepteur une méthodologie pour l'aider dans la conception du schéma conceptuel de la base de données.

- Avec le modèle relationnel, il est possible de représenter directement un schéma conceptuel de la base de données sous forme de tables représentant chacune une relation.
  - Une telle représentation ne permet pas de distinguer facilement une entité d'une association, ni de servir de support de communication si on a à discuter le schéma conceptuel avec d'autres personnes.
- Généralement, on préfère utiliser un autre formalisme tel que celui du modèle entité-association offrant une aide au niveau de la phase de modélisation

  même si cela demande un travail supplémentaire de transformation de ce modèle vers le modèle relationnel.

problèmes posés par un schéma mal conçu

- Supposons que nous avons à concevoir le schéma d'une base de données devant servir à **mettre en place une application de gestion des prêts au sein d'une bibliothèque universitaire**.
- On suppose qu'on a décidé de modéliser cette réalité par une **seule entité** que nous traduisons dans le formalisme du modèle relationnel à l'aide d'une seule relation ayant pour schéma :

PRET (Lecteur, Nom, Livre, Titre, Auteur, DatePrêt).

problèmes posés par un schéma mal conçu

Considérons maintenant l'extension suivante de la relation PRET :

Lecteur	Nom	Livre	Titre	Auteur	DatePrêt
LEC025	BENALI	L0010	Programmer en c++	M. DESMADRIL	25/12/1997
LEC032	BENOMAR	L0220	Les misérables	V. HUGO	01/10/1996
LEC025	BENALI	L1500	Introduction au PASCAL	J. DE LAGARDE	30/12/1997
LEC070	BENKADA	?????	?????	?????	?????
?????	?????	L3000	La synthèse d'images	F. MARTINEZ	?????
•••••••	•••••••	•••••••	•••••••••	•••••••••	•••••••••

On remarque que :

- Il existe des **redondances dans les données** contenues dans cette table.
  - **Chaque lecteur est répété autant de fois qu'il a emprunté de livres ce qui pose des problèmes de mise à jour**
  - **si on doit modifier la valeur d'un attribut tel que le numéro de lecteur, le nom ou l'adresse, il faudra le faire dans tous les tuples associés à ce lecteur.**
  - **C'est le cas des tuples dont la valeur de l'attribut Lecteur est égale à LEC025.**

On remarque que :

- Il faut autoriser la présence de tuples à moitié vides correspondant aux lecteurs n'ayant pas empruntés de livres et aux livres n'ayant pas été empruntés.
  - Pour de tels tuples, il sera très difficile sinon impossible **de choisir des valeurs par défaut** pour compléter celles associées aux attributs non connus (symbolisés par ??????).
  - C'est le cas du quatrième et du cinquième tuple qui correspondent respectivement au cas d'un lecteur n'ayant pas emprunté de livres et au cas d'un livre n'ayant pas encore été emprunté.

Ainsi :

- Une **mauvaise perception de la réalité** à modéliser peut conduire à un schéma mal conçu composé de relations qui rendent difficile leur manipulation aussi bien pour le SGBD que pour l'utilisateur.
- **Dans l'exemple Il aurait peut être fallu au moins trois relations :**

  une modélisant un lecteur,

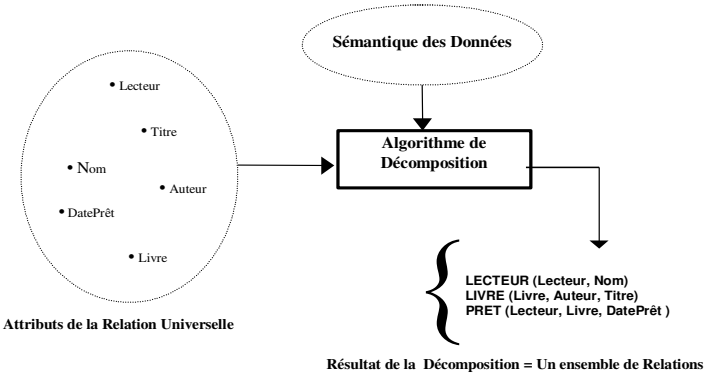
  une un livre

  et la troisième l'association entre un lecteur et un livre qui traduit l'opération d'emprunt.

L'approche de modélisation par décomposition

- L'approche par décomposition est une approche méthodologique permettant d'aboutir à un schéma conceptuel ou relationnel **qu'on peut qualifier de schéma acceptable.**
- Cette approche consiste à partir d'une relation composée de tous les attributs qu'on appelle aussi **relation universelle** et à la décomposer en plusieurs autres relations qui ne poseraient plus de problèmes.
- Cette décomposition est réalisée par **application d'un algorithme** qui nécessite en entrée les liens sémantiques qui existent entre les attributs et qu'on appelle **dépendances** entre attributs (**fonctionnelles ou autres**).
- Ce processus de décomposition est schématisé par la figure suivante :

L'approche de modélisation par décomposition



Définition d'une décomposition

- la décomposition d'une relation **R(A1 , A2 , A3 ,..... An)** peut être définie comme étant **le remplacement de cette relation par un ensemble de relations R1 , R2 ,..... Rm**
  - **dont chacune est obtenue par application d'une opération de** projection **à la relation** R
  - **et telle que la** jointure naturelle **de ces relations donne comme résultat une relation ayant le même schéma que** R.
  - ⇒ Ceci revient à dire que **l'union des attributs** des **différentes Ri** est un ensemble d'attributs **égal à (A1 , A2 , A3 ,..... An) .**
- Les opérations de **projection** et de **jointure** sont deux opérations très importantes dans les langages de manipulation de données relationnels.

1 Projection

- La projection est une opération qui consiste à supprimer des attributs d'une relation **R(A1 , A2 , ..., An)** et à supprimer les tuples en double qui peuvent apparaître dans la relation résultant de cette opération.
- Le résultat de cette opération sera donc une nouvelle relation qui possède son propre schéma (nom et liste d'attributs) et sa propre extension (ensemble de tuples).
- On note cette opération par : Π Ai1 , Ai2 , ... , Aik **(R)** qui signifie que la nouvelle relation aura pour attributs l'ensemble **Ai1 , Ai2 , ... , Aik**.
- Il s'agira donc de supprimer de R tous les attributs qui n'appartiennent pas à cet ensemble et à supprimer ensuite les tuples en double qui apparaîtront dans la nouvelle relation.
- Il faut remarquer que dans l'ensemble d'attributs **Ai1 , Ai2 , ... , Aik** , chaque attribut n'apparaît qu'une seule fois.

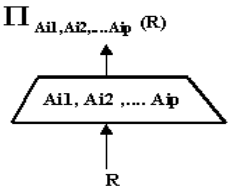
Exemple : Projection R' = Π A , C (R)

- **Soit la relation R (A,B,C)**
- **R' = Π A , C (R)** aura pour schéma **R'(A,C)**
- On élimine les valeurs des attributs B (la colonne B de la relation R) car l'attribut B n'appartient pas a R')

- et pour extension

A	B	C
a1	b1	c1
a2	b2	c2
a3	b3	c3

Relation R



Notation Graphique

A	C
a1	c1
a2	c2
a3	c3

résultat : R' (A,C)

2 Jointure naturelle

- La jointure naturelle de deux relations R et S dont les schémas ne sont pas disjoints (R et S ont au moins un attribut commun i.e. de même nom)
- est une relation T ayant pour attributs l'union des attributs de R et S et pour tuples l'ensemble des tuples obtenus par concaténation des tuples de R et S ayant les mêmes valeurs pour les attributs de même nom
- **Notation : JOIN (R,S)** ou **R ⋈ S**

☞ **L'opération de jointure naturelle est une opération associative et commutative.**

Exemple : Jointure Naturelle entre 2 relations R et S

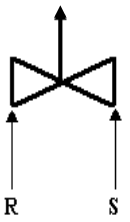
Soient les 2 relations suivantes

A	B	C
a1	b1	c1
a2	b1	c1
a3	b3	c3

Relation R

B	C	D
b1	c1	d1
b1	c1	d2
b3	c3	d3

Relation S



Notation graphique

La jointure naturelle de R et S sera une relation **R'** ayant pour schéma :

R' [(A,B,C) ∪ (B,C,D)] c.a.d. **R'(A,B,C,D)**

Les attributs de même noms dans R et S sont B et C.

Donc **la condition c** pour réaliser **l'equijointure** (*égalité des attributs de même nom*) sera :

$$R.B = S.B \wedge R.C = S.C .$$

Le résultat de la jointure naturelle de R et S

Relation R' = JOIN (R,S)

A	B	C	D
a1	b1	c1	d1
a1	b1	c1	d2
a2	b1	c1	d1
a2	b1	c1	d2
a3	b3	c3	d3

Qualité d'une décomposition

- La décomposition d'une relation **R** consiste à ***remplacer cette relation par un ensemble de relations*** **R1 , R2 , ... Rm**
- La **jointure naturelle** de ces relations doit donner comme résultat une relation **R'** **ayant le même schéma que R.**

⇒ **une relation peut être décomposée de différentes manières.**

➢ Il serait alors intéressant de savoir si **R** et **R'** ont aussi la même extension.

➢ Dans ce cas, si on avait à choisir parmi toutes les décompositions possibles de **R**, **il serait préférable de choisir celle(s) dont la jointure naturelle des relations constituant la décomposition donne une relation**

- ayant le même schéma que R
- et la même extension.

- C'est ainsi qu'on qualifie une décomposition **de décomposition sans perte** d'information par opposition à une autre qui se fait avec perte d'information.

Décomposition sans perte d'information

➢ Une décomposition d'une relation **R** en **R1 , R2 , ...; Rm** sera dite sans perte d'information si et seulement si :

1 **R1** ⋈ **R2** - - - ⋈ **Rm** **A le même schéma que R**

2 ∀ l'extension de **R**

(**R1** ⋈ **R2** - - - ⋈ **Rm**) A la même extension que **R**

Décomposition avec perte d'information

- une décomposition sera qualifiée de décomposition avec perte d'information

si l'extension de la relation résultant de la jointure naturelle de R1 , R2 , ..., Rm n'est pas égale à celle de R.

i.e il y a soit des tuples en moins dans R' soit des tuples en plus

2 Dépendances fonctionnelles

- Les dépendances fonctionnelles servent à exprimer des propriétés entre les attributs d'une relation que l'on souhaite maintenir vraies durant toute la durée de vie de la base de données.

Définition

- Etant donnée une relation **R** ayant pour schéma **R (A1, A2, ..., An)** et soient **X** et **Y** des sous-ensembles d'attributs de **(A1, A2, ..., An)** ,
- on dit que **X** $\longrightarrow$ **Y** (X détermine Y ou bien Y dépend fonctionnellement de X)
- si pour **toute extension de R**, et quelque soient les tuples **t1** et **t2** de cette extension on a:
  - **Si** $\prod$ **X (t1) =** $\prod$ **X (t2)** $\Rightarrow$ $\prod$ **Y (t1) =** $\prod$ **Y (t2)**

$\prod$ **est le symbole de l'opération de projection**

Les dépendances fonctionnelles sont des assertions sur le monde réel qu'on tente de modéliser.

- Elles définissent les propriétés que doivent vérifier les attributs entre eux

⇒ il est donc incorrecte de vouloir les déduire à partir d'une extension de la relation.

Propriétés des dépendances fonctionnelles

- Les dépendances fonctionnelles possèdent trois propriétés fondamentales qu'on appelle couramment **les axiomes d'Armstrong** car mises en évidence par W. Armstrong.

- En supposant que **X**, **Y** , **Z**, sont trois ensembles d'attributs, il est utile de faire les remarques suivantes avant d'énoncer les propriétés :

- En supposant que **X**, **Y** , **Z**, sont trois ensembles d'attributs, il est utile de faire les remarques suivantes avant d'énoncer les propriétés :

  - **L'écriture X , Z** est une écriture simplifiée de l'union ensembliste des deux ensembles d'attributs X **et** Z**.**

  ⇒ Ceci veut dire que l'écriture **X , Z** devra être interprétée comme **X ∪ Z .**

  - L'union ensembliste est un opérateur **associatif et commutatif** :

    - $((X \cup Y) \cup Z) = (X \cup (Y \cup Z))$
    - $X \cup Z = Z \cup X$

1- Réflexivité

Si $X \subseteq Y \Rightarrow Y \longrightarrow X$

Exemple :

$A \subseteq A,B \Rightarrow A,B \longrightarrow A$

$B \subseteq A,B \Rightarrow A,B \longrightarrow B$

$Nom \subseteq Nom,Prénom \Rightarrow Nom,Prénom \longrightarrow Nom$

$Prénom \subseteq Nom,Prénom \Rightarrow Nom,Prénom \longrightarrow Prénom$

2-Transitivité

Si $X \longrightarrow Y$ et $Y \longrightarrow Z \Rightarrow X \longrightarrow Z$

Exemple :

$A,B \longrightarrow C$ et $C \longrightarrow D \Rightarrow A,B \longrightarrow D$

Nom,Prénom ⟶ Tél et Tél ⟶ Adresse ⇒

Nom,Prénom ⟶ Adresse

3- Augmentation

Si $X \longrightarrow Y \Rightarrow \forall Z \quad X, Z \longrightarrow Y, Z$

Exemple :

$A,B \longrightarrow C \Rightarrow A,B,D \longrightarrow C,D$

Nom,Prénom ⟶ Age ⇒

Nom,Prénom,Tél ⟶ Age,Tél

- A partir de ces **3 axiomes** de base, on peut **déduire** facilement **d'autres propriétés** très utiles en pratiques.

- **Les plus remarquables sont :**

4- Union des parties droites

Si $X \longrightarrow Y$ et $X \longrightarrow Z \Rightarrow X \longrightarrow Y, Z$

Exemple :

$A \longrightarrow C$ **et** $A \longrightarrow D \Rightarrow A \longrightarrow C,D$

Nom,Prénom $\longrightarrow$ Age **et** Nom,Prénom $\longrightarrow$ Tél

$\Rightarrow$

Nom,Prénom $\longrightarrow$ Age,Tél

5- Décomposition

Si $X \longrightarrow Y, Z \Rightarrow X \longrightarrow Y$ et $X \longrightarrow Z$

Exemple :

$A \longrightarrow C,D \Rightarrow A \longrightarrow C$ et $A \longrightarrow D$

Numéro $\longrightarrow$ Nom,Prénom $\Rightarrow$

Numéro $\longrightarrow$ Nom et Numéro $\longrightarrow$ Prénom

6- Pseudo Transitivité

Si $X \longrightarrow Y$ et $Y, W \longrightarrow Z \Rightarrow X, W \longrightarrow Z$

Exemple :

$A \longrightarrow C$ et $B,C \longrightarrow D \Rightarrow A,B \longrightarrow D$

Nom,Prénom $\longrightarrow$ Adresse et

Tél, Adresse $\longrightarrow$ Père $\Rightarrow$

Nom,Prénom,Tél $\longrightarrow$ Père

7- Distributivité par rapport à l'union

Si $X \longrightarrow U \Rightarrow \forall\ u_i \subseteq U\ X \longrightarrow ui$

c'est une généralisation de la décomposition

Exemple :

$A \longrightarrow C,D,E,F \Rightarrow$

$A \longrightarrow C$ et $A \longrightarrow D$ et $A \longrightarrow E$ et $A \longrightarrow F$

$A,B \longrightarrow C,D,E \Rightarrow$

$A,B \longrightarrow C$ et $A,B \longrightarrow D$ et $A,B \longrightarrow E$

Les propriétés 4 5 6 et 7 peuvent être démontrées à partir des axiomes d'Armstrong

Dépendance fonctionnelle élémentaire

- une dépendance de la forme : $X \longrightarrow A$

  ou

  - **A** est un **attribut unique non inclus** dans **X** **($A \not\subset X$)**

  - $\forall X' \subset X$, **il n'existe pas de d.f.** $X' \longrightarrow A$.

$\Rightarrow$ une d.f. est élémentaire si elle n'est pas obtenue **par augmentation** de sa partie gauche.

exemple :

Si on considère les deux D.F. :

(NUMERO, NOM) $\longrightarrow$ ADRESSE (1)

NUMERO $\longrightarrow$ ADRESSE (2)

- la D.F. **(NUMERO, NOM) $\longrightarrow$ ADRESSE** est une D.F. **non élémentaire**

  car NUMERO $\subset$ (NUMERO,NOM)

  et il existe la D.F. **NUMERO $\longrightarrow$ ADRESSE.**

- Par contre la D.F. **NUMERO $\longrightarrow$ ADRESSE est une D.F. élémentaire** car elle vérifie les propriétés d'une D.F. élémentaire.

Dépendance fonctionnelle transitive

- **une d.f. de la forme : $X \longrightarrow A$**

  ou

  - **A** est un attribut unique non inclus dans X ($A \not\subset X$),
  - $\exists\ Y \not\subset X,\ X \longrightarrow Y$ et $Y \longrightarrow A$ et $Y \not\longrightarrow X$

⇒ Dans ce cas, on dit aussi que **A est transitivement dépendant de X**.

✎ Dans le cas où **A n'est pas** transitivement dépendant de X, on dit que ***A est directement dépendant de X*** (d.f. directe).

exemple :

- **Si on considère les D.F. :**
- **① $A \longrightarrow B$ ② $B \longrightarrow C$ ③ $A \longrightarrow C$**
- **④ $D \longrightarrow B$ ⑤ $D \longrightarrow C$ ⑥ $B \longrightarrow D$**

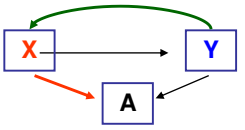
- La D.F. **$A \longrightarrow C$ est transitive car**
  - $\exists\ B \not\subset A$ tel que $A \longrightarrow B$ et $B \longrightarrow C$ et $B \not\longrightarrow A$
- La D.F. **$D \longrightarrow C$ n'est pas transitive car**
  - $\exists\ B \not\subset D$ tel que $D \longrightarrow B$ et $B \longrightarrow C$
  - **mais $B \longrightarrow D$**

Dépendance fonctionnelle directe

- une d.f de la forme : $X \longrightarrow A$

ou :

- A est un attribut unique non inclus dans X ($A \not\subset X$)
- $\forall Y \not\subset X$, si $X \longrightarrow Y$ et $Y \longrightarrow A$ alors la d.f. $Y \longrightarrow X$ existe aussi



exemple :

Si on considère les D.F. :

① $A \longrightarrow B$ ② $B \longrightarrow C$ ③ $A \longrightarrow C$

④ $D \longrightarrow B$ ⑤ $D \longrightarrow C$ ⑥ $B \longrightarrow D$

- La D.F. $A \longrightarrow C$ n'est pas directe car pour $B \not\subset A$ on a :
  - **$A \longrightarrow B$ et $B \longrightarrow C$ mais $B \longrightarrow A$ n'existe pas**
- La D.F. $D \longrightarrow C$ **est directe** puisque pour $B \not\subset D$ on a :
  - **$D \longrightarrow B$ et $B \longrightarrow C$ et on a bien $B \longrightarrow D$**

- La D.F. A ⟶B est directe puisque le seul attribut Y ⊄ A pour lequel on a une d.f. A ⟶Y est C

  En effet on a la d.f. A ⟶ C mais on n'a aucune d.f. C ⟶ B pour vérifier si C ⟶ A existe ou non

  Donc la d.f. **A ⟶B est forcément directe** car pas de contradiction avec la définition

- Dans le cas où **A n'est pas directement** dépendant de X on dit que A est transitivement dépendant de X.

Dépendance fonctionnelle totale, pleine ou complète

- **une d.f. de la forme X ⟶ A**
- **ou**
  - A est un attribut unique non inclus dans X (A ⊄ X)
  - **∀ X' ⊂ X , il n'existe pas de d.f X' ⟶ A .**

⇒ En d'autres termes, la dépendance **X ⟶ A est élémentaire.**

- on dit aussi que **A est pleinement** ou totalement ou complètement dépendant de **X**.
- Dans le cas contraire, on dit que **A** est partiellement dépendant de **X**
  - ✓ **i.e. A dépend d'une partie de X**
  - ✓ **donc forcément de X aussi (par augmentation)**

exemple :

- **Si on considère les D.F. :**

  ① A ,B ⟶ C ② B ⟶ C ③ B,C ⟶ D

- dans la d.f. **A ,B ⟶ C**
  - **C n'est pas totalement dépendant de A ,B** car
  - **on a B ⊂ A,B tel que B ⟶ C.**

  ⇒ **C est partiellement dépendant de A,B.**

- **dans la d.f. B,C ⟶ D**
  - **D est totalement dépendant** de B,C car
  - il n'existe pas de **X' ⊂ B ,C** et tel que X' ⟶ D.
  - i.e. **B** ⟶/ D et **C** ⟶/ D.

Dépendance fonctionnelle triviale

- une d.f. de la forme **X ⟶ Y** , tel que Y ⊆ X.
- En d'autres termes, une d.f. est triviale si elle est **obtenue grâce** à la propriété de réflexivité.

exemple :

Si on considère les D.F. :

① A ,B ⟶ B ② B,C ⟶ C ③ A ,C ⟶ D

- la d.f. ① **est triviale** car on a **B ⊂ A ,B**
- la d.f. ② **est triviale** car on a **C ⊂ B ,C**
- la d.f. ③ **n'est pas triviale** car on a **D ⊄ A ,C**

❖ les notions de :

- **d.f. élémentaire**
- **D.f totale**
- **D.f. directe**

interviennent dans la définition des **formes normales** d'une relation et principalement la 2FN et la 3FN.

Fermeture d'un ensemble de d.f.

- Le fermeture d'un ensemble de d.f. **F** et qu'on note **F^+** s'obtient à partir de F par **application des propriétés** des d.f. qui permettent de générer d'autres d.f.
  - **conduit à** augmenter **progressivement** F **avec de nouvelles d.f.**
  - à partir d'un certain moment, l'ensemble obtenu **va devenir stationnaire : on dit que l'on a obtenu la fermeture F^+ de F.**

- les **nouvelles d.f** obtenues par applications des propriétés sont en fait **redondantes** puisqu'elles s'obtiennent à partir d'autres.

  ⇒ Le problème va consister donc à **éliminer ce type de d.f.** afin de ne conserver qu'un ensemble minimal de d.f représentant la même sémantique que l'ensemble initial mais sans redondance.

  - Cet ensemble s'appelle une **couverture minimale de l'ensemble initial F.**

- la fermeture F^+ peut comporter beaucoup plus de d.f. que F et plus le nombre de d.f. de F est grand plus le calcul de F^+ sera long.

  - **En pratique, on ne sera presque jamais amené à calculer l'ensemble F^+**

Equivalence entre 2 ensembles de d.f.

➢ 2 ensemble **F** et **G** de d.f. sont équivalents si et seulement si la **fermeture** de **F** est **égale** à celle de **G** :

$$F^+ = G^+$$

Couverture minimale d'un ensemble de d.f

- notée $\mathbf{C_F}$
- un ensemble de d.f. équivalent à **F** : $\mathbf{C_F}^{+} = \mathbf{F}^{+}$
- **ayant les propriétés suivantes :**

1- **Toute d.f.** de $\mathbf{C_F}$ à une **partie droite** composée d'un seul attribut

2- $\forall\ \mathbf{X} \longrightarrow \mathbf{A} \in \mathbf{C_F}$ et $\mathbf{Z} \subset \mathbf{X}$,

$\{(\mathbf{C_F} - \mathbf{X} \longrightarrow \mathbf{A}) \cup \mathbf{Z} \longrightarrow \mathbf{A}\}$ **n'est pas équivalent** à $\mathbf{C_F}$

3- $\forall\ \mathbf{X} \longrightarrow \mathbf{A} \in \mathbf{C_F}$, $\mathbf{C_F} - (\mathbf{X} \longrightarrow \mathbf{A})$ **n'est pas équivalent** à $\mathbf{C_F}$

- **La propriété (1)** peut s'obtenir en décomposant toute **d.f.** dont la partie droite n'est pas unique grâce à l'axiome de décomposition.
  - **Ceci permet de manipuler des** d.f. **ayant une partie droite simple.**
- **La propriété (2)** permet de garantir qu'aucun attribut dans la partie gauche d'une D.F. de **CF** n'est redondant.
  - **Elle permet de supprimer les** d.f. **obtenues par application de l'axiome d'augmentation.**
- **La propriété (3)** permet de garantir **qu'aucune d.f.** n'est redondante dans **CF**

- Tout ensemble de d.f. **F** possède **au moins une couverture** minimale ayant les propriétés citées plus haut.
- Une telle couverture **s'obtient par construction** à partir de **F** en satisfaisant les propriétés indiquées plus haut.
- Elle sert principalement comme entrée à l'algorithme de décomposition en troisième forme normale

Fermeture d'un ensemble d'attributs

- Lors de la décomposition d'une relation vérifiant un ensemble de d.f., il est important de pouvoir vérifier si cette décomposition préserve les d.f.
- Pour cela, il faudra impérativement calculer la fermeture **F+** de l'ensemble **F** des dépendances vérifiées par la relation décomposée.
- Plus le nombre de dépendances dans **F** est grand, plus le temps de calcul de **F+** sera grand.
- Il existe une autre méthode pour vérifier si une d.f. $\mathbf{X} \longrightarrow \mathbf{Y} \in$ ou non à **F+** sans passer par le calcul de **F+** .

Cette méthode est justement basée sur la notion de fermeture d'un ensemble d'attributs.

- Formellement, la fermeture d'un ensemble d'attributs **X** et qu'on note X^+
  - **est un ensemble d'attributs**
  - **tel que** : $X^+ = \{ \bigcup Ai \ / \ X \longrightarrow Ai$ **peut être déduite de F par application des axiomes d'Armstrong** $\}$.

☞ L'intérêt pratique de calculer X^+ est donc de pouvoir vérifier facilement si $X \longrightarrow Y \in$ ou non à F^+ sans être amené à calculer F^+

Lemme

- Une dépendance fonctionnelle $V \longrightarrow U \in F^+$ **si et seulement si** $U \subseteq V^+$

  où
  - V^+ est la fermeture de V relativement à l'ensemble de dépendances fonctionnelles F
  - V^+ contient tous les attributs A tel que la d.f. $V \longrightarrow A$ peut être déduite de F par application des axiomes d'Armstrong.

❖ Ce lemme peut être utilisé **pour montrer qu'une D.F. est redondante** dans un ensemble F de dépendances fonctionnelles

Algorithme de calcul de X^+

```
Fermeture (X)
    $ entrée :
        X un ensemble d'attributs
        F un ensemble de dépendances fonctionnelles
    $ Sortie :
        X+ : Fermeture de X par rapport à F
    Début
    |   X+ ⟵ X ;
    |   Fin ⟵ Faux;
    |
    | Tant que  ⌉Fin  faire
    | |
    | |   Temp ⟵ X+ ;
    | |
    | |   Pour  toute dépendance fonctionnelle Y ⟶ Z de F  Faire
    | |   |
    | |   |     Si   Y ⊆ X+
    | |   |     alors     X+ ⟵ X+ ∪ Z;
    | |   |     Fin si
    | |   Fin Pour
    | |   Si   Temp = X+   Alors   Fin ⟵ Vrai ;
    | Fin Tan que
    Fin
Fin Fermeture
```


exemple :

Soit l'ensemble F de d.f. :

F = {

AB	⟶	C ;	①
C	⟶	A ;	②
BC	⟶	D ;	③
ACD	⟶	B;	④
D	⟶	EG ;	⑤
BE	⟶	C;	⑥
CG	⟶	BD ;	⑦
CE	⟶	AG	⑧

}

- Calculez la fermeture $(BD)^+$ de l'ensemble d'attributs (BD)

étape 0 : $(BD)^+ = \{BD\}$

étape 1 : $(BD)^+ = \{BDEG\}$ car dans ⑤ on a $D \subseteq (BD)^+$.

on ajoute EG à (BD) +

étape 2 : $(BD)^+ = \{BDEGC\}$ d'après ⑥

étape 3 : $(BD)^+ = \{BDEGCA\}$ d'après ⑧

- on remarque que $(BD)^+$ contient tous les attributs.
- Donc, on s'arrête car quelque soit la d.f. à examiner et l'attribut à rajouter ça ne changera pas $(BD)^+$.
- On est dans le critère d'arrêt de l'algorithme TEMP = $(BD)^+$

Vérifier si la d.f. **CG ⟶ B est redondante dans F**

- on calcule la fermeture (CG) + de CG relativement à { F - CG ⟶ B }
- et on voit si $B \subseteq CG+$

On va donc travailler avec F'= F - CG ⟶ B

étape 0 : (CG) + = {CG}

étape 1 : (CG) + = {CGD} grâce à la d.f. CG ⟶ D

étape 2 : (CG) + = {CGDA} d'après ② C ⟶ A

étape 3 : (CG) + = { CGDAB } d'après ④ ACD ⟶ B

On remarque qu'après cette étape $B \subseteq CG+$.

On peut s'arrêter et conclure que la d.f. CG ⟶ B est redondante dans F.

Pour vérifier ce résultat, il suffit de montrer qu'on peut la déduire de F' par applications des propriétés des d.f.

En effet :

1. C ⟶ A ⇒ CG ⟶ AG **augmentation avec *G***
2. CG ⟶ AG ⇒ CG ⟶ ACG **augmentation avec *C***
3. CG ⟶ D et CG ⟶ ACG ⇒ CG ⟶ ACDG **union des parties droites**
4. CG ⟶ ACDG ⇒ CG ⟶ ACD et CG ⟶ G **décomposition**
5. CG ⟶ ACD et ACD ⟶ B ⇒ CG ⟶ B **transitivité (cqfd)**

Définition formelle d'une clé

- La clé d'une relation **R(A1 ,A2 , ..., An)** est un sous ensemble d'attributs **X** tel que :

  1- **X ⊆ (A1 ,A2 , ..., An)**

  2- **X ⟶ A1 ,A2 , ..., An (i.e. X ⟶ A1 ; X ⟶ A2 ; ..... ; X ⟶ An)**

  3- **Il n'existe pas Y ⊂ X tel que : Y ⟶ A1 ,A2 , ..., An**

- une relation peut posséder plusieurs clés qui satisfont les propriétés ci-dessus.
- Ce sont les **clés candidates** de la relation.

Démarche de recherche des clefs candidates

- Connaissant les dépendances fonctionnelles vérifiées par une relation **R(Ω)**, les clés candidates peuvent facilement se démontrer en utilisant les propriétés des d.f.
- L'algorithme de calcul de la fermeture d'un ensemble d'attributs peut servir comme base pour la recherche des clefs.

➢ La connaissance des clés d'une relation est primordiale car leur connaissance ainsi que celle des d.f. vérifiées par la relation permettent :

- **d'une part de déterminer la forme normale de la relation**
- **et d'autre part d'envisager une décomposition de la relation si celle-ci n'est pas dans une forme normale convenable.**

Objectifs de la normalisation

- La conception d'un schéma relationnel ne doit pas être faite d'une façon quelconque
- doit être soumise à une normalisation dont l'objectif est de permettre l'obtention de relations vérifiant certaines propriétés telles que :
  - Faciliter les opérations de création , de suppression et de mise à jour d'un tuple
  - Faciliter la modification d'un schéma
  - Permettre l'utilisation d'algorithmes efficaces pour la recherche d'un tuple
  - Rendre possible la représentation d'une relation quelconque

Les deux tendances de définition des formes normales

- Il existe 2 tendances totalement différentes qui conduisent le plus souvent a des énoncés vrais dans un cas mais faux dans l'autre.

➢ **La première tendance** vise a définir la forme normale d'une relation en **tenant compte du fait que la clef primaire** de la relation **est déjà choisie** parmi les clefs candidates éventuelles.

➢ Ceci veut dire que si la relation est dans une certaine forme normale(2ième, 3ième, BCNF), elle ne l'est que relativement à la clef primaire retenue.

☝ Ceci sous entend aussi que la forme normale trouvée pour cette relation peut ne pas être vraie relativement à une autre clef candidate de cette relation.

➢ Cette tendance impose donc comme contrainte au moment de la normalisation d'une relation de choisir d'abord une clef primaire **K**.

➢ De plus ce choix devra aussi être respecté au moment de l'implémentation en spécifiant lors de la création de la relation (grâce au LDD) que sa clef primaire est **K**.

➢ La deuxième tendance est plus généraliste en ce sens qu'elle n'impose pas au moment de la normalisation le choix d'une clef primaire.

➢ Ainsi, si la relation est dans une certaine forme normale(2ième, 3ième, BCNF), elle le sera quelle que soit la clef primaire qui sera choisie parmi toutes les clefs candidates de la relation.

➢ Ceci veut dire qu'au moment de l'implémentation de la relation on peut retenir n'importe quelle clef candidate comme clef primaire et ceci ne remettra pas en cause la forme normale trouvée.

➢ cette deuxième tendance est à privilégier car elle est plus générale et sépare clairement entre les étapes de conception d'un schéma relationnel et de son implémentation.

La première forme normale (1FN)

- La première forme normale ne se base pas sur la notion de clef ni de d.f. Elle a pour objectif de rendre possible la représentation sous forme de table d'une relation.

- Une relation sera dite **en 1FN** si chaque attribut de cette relation a un domaine de valeurs simple c'est à dire les valeurs que peut prendre chaque attribut sont des **valeurs atomiques** (i.e. non décomposables)

Exemple :

- Soit la relation **AVION (TYPEAVION, CAPACITE)**
- l'attribut **TYPEAVION** désigne le type d'un avion
- l'attribut **CAPACITE** désigne le nombre de places que peut contenir un avion de ce type.

Sachant que dans la réalité, il peut exister plusieurs modèles d'un même type d'avion chacun avec une capacité ;

un exemple d'extension d'une telle relation serait donc :

TYPEAVION	CAPACITE
CARAVELLE	(100)
CONCORDE	(400, 600)
AIRBUS	(250, 275)
B707	(180, 150)
B747	(350)

- L'attribut **CAPACITE** est donc un attribut qui n'est pas atomique. Il possède des valeurs qui peuvent être décomposées en d'autres valeurs qui sont atomiques.
-
- Par exemple la valeur (400, 600) de l'attribut CAPACITE qui figure dans le deuxième tuple peut être décomposée en deux valeurs atomiques (indécomposables) qui sont 400 et 600.
- Cette relation **n'est donc pas en 1FN**.
- On dit aussi qu'elle **n'est pas normalisée**.

Pour la mettre en 1FN, on peut :

a) éclater le groupe répétitif CAPACITE en créant un nouveau tuple pour chaque valeur différente du groupe. On aura donc:

TYPEAVION	CAPACITE
CARAVELLE	100
CONCORDE	400
CONCORDE	600
AIRBUS	250
AIRBUS	275
B707	180
B707	150
B747	350

☝ **L'inconvénient** de cette solution est la **redondance** des autres attributs car pour chaque valeur du groupe on crée un nouveau tuple avec les mêmes valeurs des attributs sauf la capacité qui change.

b) Une autre solution serait **si on connaît la cardinalité maximum** des valeurs **du groupe**, de **créer** pour chaque valeur une **colonne** dans la relation **AVION1**.

Dans notre exemple, on sait que la cardinalité maximum des valeurs du groupe est 3.

Donc on crée dans **AVION1** trois nouvelles colonnes qu'on appellera par exemple **CAP1, CAP2** et **CAP3** et dans chaque colonne on mettra une valeur de **CAPACITE** prise dans le groupe. Ceci donnera la relation suivante :

TYPEAVION	CAP1	CAP2	CAP3
CARAVELLE	100	?	?
CONCORDE	400	600	?
AIRBUS	250	275	?
B707	180	150	?
B747	350	?	?

ou le **?** symbolise une valeur indéfinie.

- L'inconvénient de cette solution est l'ajout des valeurs indéfinies ou NULL qui complique la gestion

☝ beaucoup de SGBD ne savent pas bien gérer les valeurs NULL.

La deuxième forme normale (2FN)

Définition 1 (première tendance)

Une relation est en **2FN** si et seulement si :

1) Elle est en 1FN

2) Tout attribut **n'appartenant pas à la clef primaire** est **pleinement dépendant** de la **clef primaire.**

Cette définition traduit le fait que toutes les dépendances ayant en partie droite un attribut **n'appartenant pas à la clef primaire** et en partie gauche la **clef primaire sont élémentaires.**

Exemple :

Soit la relation

R (NUMETUDIANT , NOM , DIPLOME , INSTITUT)

ayant deux clefs candidates qui sont **NUMETUDIANT** et **(NOM,DIPLOME).**

On suppose aussi qu'elle vérifie la D.F. **:**

DIPLOME ⟶ INSTITUT:

Dans le cas où on retient la clef candidate **NUMETUDIANT comme clef primaire,**

☞ cette relation **est bien en 2FN** puisque tous les attributs n'appartenant pas à cette clef primaire sont en dépendance totale de cette même clef :

NUMETUDIANT ⟶INSTITUT ;

NUMETUDIANT ⟶ NOM ;

NUMETUDIANT ⟶ DIPLOME

Par contre si on avait retenu comme **clef primaire** l'autre clef candidate **(NOM,DIPLOME),**

⇒ cette relation **n'aurait pas été en 2FN**

car un attribut **INSTITUT** n'appartenant pas à cette clef primaire n'est pas en dépendance totale de cette clef primaire puisqu'il dépend **d'une partie DIPLOME** de cette clef :

DIPLOME ⟶INSTITUT

☝ avec cette première tendance **il faut être très attentif** en précisant bien que la forme normale est fonction de telle ou telle clef primaire.

Exemple 2 :

Soit la relation **STOCK (PROD, DEPOT, LIBELLE, QTE)** ou les attributs ont les sens suivants :

(**PROD** : *numéro du produit* ; **DEPOT :** *numéro du dépôt ou est stocké le produit* ; **LIBELLE:** *nom du produit* ; **QTE :** *quantité du produit stockée dans le dépôt*)

On suppose que la clé de cette relation est formée par le couple d'attributs **(PROD, DEPOT)**

et que cette relation vérifie les D.F. suivantes :

{ **(PROD,DEPOT) ⟶ QTE** ; **PROD ⟶ LIBELLE** }

- Cette relation **est en 1FN** car tous ses attributs possèdent des valeurs **atomiques** (ou du moins on l'admet en l'absence d'une extension qui pourrait contredire ce fait).

- **Elle n'est pas en 2FN** à cause de la D.F. **PROD ⟶ LIBELLE**

  qui a dans la partie droite un attribut **(LIBELLE)** ∉ à **la clé**,

  et qui a dans la partie gauche un attribut **: PROD**, qui constitue une **partie de cette clé**.

- Toute relation dont **la ou les clé(s)** est (sont) **composée**(s) **d'un seul attribut, est forcément en 2FN**

  car il n y aura jamais de D.F. ayant en partie gauche un sous ensemble d'attribut contenus dans la ou les clé(s) de cette relation

La deuxième forme normale (2FN)

Définition 2 (deuxième tendance)

Une relation est en 2FN si et seulement si :

1) Elle est en 1FN

2) Tout attribut **non primaire** est **pleinement dépendant de chaque clef candidate**

cette définition est plus générale et ne se restreint pas uniquement à la notion de clef primaire.

Elle traduit le fait que **toutes les d.f.** ayant en partie droite un attribut **non primaire** et en partie gauche **une clef candidate sont élémentaires.**

Exemple :

Si on reprend l'exemple de la relation **R (NUMETUDIANT , NOM , DIPLOME , INSTITUT)**,

les attributs primaires sont :

NUMETUDIANT , **NOM et DIPLOME.**

Le seul attribut **non primaire** (i.e. qui n'appartient à aucune clef candidate) est **INSTITUT.**

Et puisque cet attribut **n'est pas en dépendance totale** de la clef candidate **(NOM,DIPLOME)** du fait qu'il **dépend d'une partie DIPLOME** de cette clef :

DIPLOME ⟶INSTITUT

On peut dire que cette relation **n'est pas en 2FN**.

⇒ *On voit que cette conclusion est générale et est valable quelle que soit la clef primaire qu'on retiendra au moment de l'implémentation.*

Problèmes posés par une relation qui n'est pas en 2FN

Exemple :

La relation **STOCK (PROD, DEPOT, LIBELLE, QTE)**

dont la clé est le couple **(PROD, DEPOT).**

n'**est pas en 2FN** a cause de la d.f. **PROD ⟶ LIBELLE.**

C'est justement cette D.F. qui cause des problèmes à la relation

Ce problèmes peuvent être résumés comme suit :

- **Insertion de tuples :**

  - Si on décide de rajouter un nouveau produit **p** ayant pour libellé **l** dans la BD, il ne sera pas possible d'insérer cette information

  - on doit connaître au moins un **Dépôt** où le stocker (une valeur **d** de l'attribut **DEPOT**) puisque la clé est le couple **(PROD, DEPOT).**

  - En effet, pour insérer un tuple dans la base **il faut que sa clé soit connue** au moment de l'insertion.

  - Or les attributs **PROD** et **LIBELLE** sont liés sémantiquement et ceci indépendamment des autres attributs, **Mais**

    on est obligé de connaître au moins une valeur de l'attribut DEPOT, pour pouvoir rajouter une information sur un produit.

- **Suppression de tuples** (ou perte d'information)**:**

  - Si parmi tous les tuples de la relation, il existe un seul tuple contenant l'information que le produit **P1** a pour libellé **Armoires**,

  - la suppression d'un tel tuple de la base va engendrer la perte de cette information.

  - La conséquence de cette suppression est qu'on ne pourra plus savoir que le produit **P1** a pour libellé **Armoires.**

- **Mise à jour de tuples :**

- Dans tous les tuples correspondant aux dépôts dans lesquels est stocké le produit **P1** , on va avoir le couple de valeur **(P1 , Armoires)** qui sera dupliqué dans chaque tuple.

- Cette redondance d'information pose des problèmes au niveau de la mise à jour.

- En effet, si on veut mettre à jour la valeur **P1** dans la colonne **PROD** ou **Armoires** dans la colonne **LIBELLE** on est obligé de le faire partout d'où un problème de perte de temps.

- Et si par souci de gain de temps, on ne fait la mise à jour qu'à un seul endroit (sur un seul tuple) on engendre un **problème d'incohérence** dans la base de données (une même donnée possède deux valeurs différentes dans la base).

- **Redondance d'information :**

- En général pour tous les tuples correspondant aux dépôts dans lesquels est stocké le produit **Pi** ayant pour libellé **Li** on va avoir le couple de valeur **(Pi , Li)** qui sera dupliqué au niveau de chaque tuple.

- C'est bien un problème de redondance d'information puisque la même valeur du couple d'attributs **(PROD , LIBELLE)** et qui est **(Pi , Li)** est stockée plusieurs fois dans la base de données.

La troisième forme normale (3FN)

Définition 1 (première tendance)

Une relation est en 3FN si et seulement si :

1) Elle est en 2FN

2) Tout attribut n'appartenant pas à la clef primaire est **directement dépendant** de la **clef primaire**.

Cette définition traduit le fait que tout attribut n'appartenant pas à la clef primaire **n'est pas transitivement dépendant** de la **clef primaire**.

➢ **Une autre définition plus condensée serait :**

Une relation est en 3FN si et seulement si tout attribut n'appartenant pas à la clef primaire **est pleinement et directement dépendant de la clef primaire**.

Exemple :

Soit la relation R (A,B,C,D)

ayant pour clef primaire le couple (A,B)

et vérifiant les D.F. suivantes : { C ⟶ A ; D ⟶ B }

les attributs n'appartenant pas à la clef primaire sont : C et D.

Ces deux attributs sont pleinement dépendant de la clef primaire car on a bien : (A,B) ⟶ C et (A,B) ⟶ D

et ces deux d.f. sont élémentaires.

⇒ **La relation est donc en 2FN.**

De même que les attributs C et D sont directement dépendant de la clef primaire

i.e. les deux d.f. (A,B) ⟶ C et (A,B) ⟶ D sont directes.

⇒ **La relation est donc aussi en 3FN.**

Définition 2 (deuxième tendance)

Une relation **est en 3FN** si et seulement si :

1) Elle est en 2FN

2) Tout attribut **non primaire** est **directement dépendant** de **chaque clef candidate** de la relation.

cette définition est plus générale et ne se restreint pas uniquement à la notion de clef primaire.

Elle traduit le fait que **toutes les dépendances** ayant en partie droite un attribut **non primaire** et en partie gauche une **clef candidate sont élémentaires et directes**.

❖ **Une autre définition plus condensée serait :**

Une relation est en 3FN si et seulement si **tout attribut non primaire** est **pleinement et directement** dépendant de chaque clef candidate de la relation.

Exemple :

Soit la relation R (A ,B, C, D)

ayant deux clefs candidates (A, B) et (B, D)

et vérifiant les D.F. suivantes : { D ⟶ B ; C ⟶ D }

les attributs primaires sont : A, B et D.

Le seul attribut non primaire est C.

La relation est en 2FN du fait que les d.f. de C sur chaque clef candidate sont élémentaires : A, B ⟶ C et B,D ⟶ C.

C est pleinement dépendant de chaque clef candidate. On peut dire aussi que :

(A,B) ⟶ C et (B,D) ⟶ C sont élémentaires.

⇒ **La relation est donc en 2FN.**

De même que l'attribut non primaire **C est directement** dépendant de chaque clef candidate

i.e. les deux d.f. (A,B) ⟶ C et (B,D) ⟶ C **sont directes**.

⇒ **La relation est donc aussi en 3FN.**

Problèmes posés par une relation qui n'est pas en 3FN

Soit la relation R (NUMETUDIANT, NOM, DIPLOME, INSTITUT).

On suppose que R a une seule clé qui est formée par l'attribut **NUMETUDIANT** et qu'elle vérifie la d.f. :

INSTITUT ⟶ DIPLOME

Cette relation **n'est pas en 3FN** à cause de la D.F. :

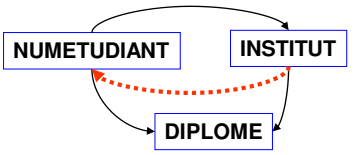
INSTITUT ⟶ DIPLOME dans laquelle un attribut non primaire : DIPLOME ∉ à la clé et **dépend transitivement** de cette clé.

En effet, d'après les d.f. :

NUMETUDIANT ⟶ INSTITUT

NUMETUDIANT ⟶ DIPLOME

INSTITUT ⟶ DIPLÔME



- on peut montrer que la d.f. NUMETUDIANT ⟶ DIPLOME **n'est pas directe**

cette d.f. serait directe si : ∀ Y ⊄ NUMETUDIANT, NUMETUDIANT ⟶ Y et Y ⟶ DIPLOME ⇒ Y ⟶ NUMETUDIANT

Or en prenant Y= INSTITUT on a bien :

NUMETUDIANT ⟶ INSTITUT et INSTITUT ⟶ DIPLOME **mais on n'a pas INSTITUT ⟶ NUMETUDIANT**

donc la d.f. NUMETUDIANT ⟶ DIPLOME **d'un attribut non primaire** sur la clef n'étant **pas directe**,

➢ cette relation **ne peut être en 3FN**.

☞ **Elle est simplement en 2FN puisqu'elle possède une clé formée d'un seul attribut.**

- La d.f. qui cause des problèmes **n'est pas la d.f.**

NUMETUDIANT ⟶ DIPLOME

Mais

INSTITUT ⟶ DIPLOME

car c'est elle qui engendre **une d.f. transitive** de DIPLOME sur la clé.

➢ On peut résumer les problèmes posés selon leur type comme suit :

➢ **Insertion de tuples :**

- Si un institut i décide de dispenser pour la première fois un diplôme d, il ne serait pas possible d'insérer cette réalité dans la base si on ne connaît pas au moins un étudiant (identifié par NUMETUD qui est la clé) qui veut suivre ce diplôme .
- Ceci pose un problème dans la mesure ou la réalité qu'un institut dispense un diplôme est indépendante de celle relative à l'inscription d'un étudiant dans ce diplôme.

➢ **Suppression de tuples** (ou perte d'information):

- Si parmi tous les tuples de la relation, il existe un seul tuple contenant l'information que l'institut i1 dispense le diplôme d1, le fait de supprimer ce tuple engendre la perte de cette information.
- On ne pourra plus savoir que le diplôme dispensé par l'institut i1 est d1 .

➢ **Mise à jour de tuples :**

- Pour tous les étudiants préparant le diplôme dm dans l'institut im le couple de valeur (im , dm) sera dupliqué pour chaque étudiant.
- C'est essentiellement un problème de redondance qui pose des problèmes de mises à jour.
- Si on veut mettre à jour la valeur im ou dm on est obligé de le faire partout (d'où un problème de perte de temps) ou le faire en un seul endroit (sur un seul tuple) et on a alors un problème d'incohérence des données (une même donnée possède **deux valeurs différentes dans la base).**

➢ **Redondance d'information :**

- En général pour tous les étudiants préparant le diplôme dm dans l'institut im le couple de valeur (im, dm) sera dupliqué pour chaque étudiant.
- C'est bien un problème de redondance d'information puisque la même valeur du couple d'attributs (INSTITUT, DIPLOME) et qui est (im, dm) est stockée plusieurs fois dans la base de données.

Forme normale de BOYCE ET CODD

- Les définitions de la 2FN et de la 3FN tenaient compte uniquement des attributs non primaires et de leur dépendances fonctionnelles vis à vis des clefs candidates.
- On s'est rendu compte que même les attributs primaires participant dans certaines d.f. pouvaient posaient des problèmes au niveau des relations.
  - **En d'autres termes, même une relation qui se trouve en 3FN peut encore poser des problèmes.**

♉ C'est pour cela qu'a été introduite cette nouvelle forme normale appelée la forme normale de BOYCE et CODD (ou BCNF)

- La définition de la BCNF ne privilégie aucune clé et s'énonce ainsi :

➢ Une relation R est en BCNF si

  – quelque doit la d.f $X \longrightarrow A$ (avec $A \not\subset X$)
  – **alors** X est une clé de R ou contient une clé de R
    (on dit aussi dans ce cas que X est une super-clé de R).

Exemple :

- **Soit la relation ADRESSE(Code_Postal, Ville, Nom_Rue)** ayant pour **clé** le couple d'attributs **(Ville, Nom_Rue)** et munie des **d.f.** suivantes :

  – **Ville, Nom_Rue ⟶ Code_Postal** ①
  – **Code_Postal ⟶ Ville** ②

- En utilisant les propriétés des d.f. **on peut déduire** de ② par **augmentation** avec l'attribut **Nom_Rue** que :

  – **Code_Postal , Nom_Rue ⟶ Ville, Nom_Rue** ③

- Ainsi, cette relation possède une autre clé candidate qui est :

  (Code_Postal , Nom_Rue)

- Comme **il n'existe pas d'attribut non primaire** (tous les attributs sont primaires),

  la relation est au moins en troisième forme normale 3FN.

- Par contre dans la d.f. **Code_Postal ⟶ Ville**

  - l'attribut **Code_Postal n'est pas une clé** pour la relation ni ne contient une clé de la relation.
  - **Cette relation n'est donc pas en BCNF.**

- On remarque que bien que la relation **ADRESSE soit en 3FN**, **le problème de redondance d'informations existe toujours.**

Exemple :

Code_Postal	Ville	Nom_Rue
C1	V1	R1
C1	V1	R2
C2	V2	R3
C2	V2	R4

- **On remarque que :**
  - on a plusieurs fois la même valeur **C1** qui apparaît dans la colonne Code_Postal,
  - plusieurs fois la même valeur **V1** qui apparaît dans la colonne Ville,
  - plusieurs fois la même valeur **V2** qui apparaît dans la colonne **Ville**

En conclusion

- Même si une relation **est en 3FN**, ceci **n'élimine pas complètement** le problème de redondance d'information

  ⇒ **d'où la nécessité de passer à la forme normale de BOYCE et CODD.**

✍ Toute relation **admet une décomposition** en BCNF

- **qui préserve la jointure**

☝ **mais qui ne préserve pas généralement les d.f.**

3 Les méthodes de conception d'un schéma

- Les méthodes de conception utilisées dans le cadre du modèle relationnel forment deux grandes familles :
  - les méthodes **agrégatives** **(ou ascendantes ou aussi par synthèse)**
  - **et les** méthodes **par décomposition** **(ou descendantes)**

➢ On dispose pour chaque famille d'un algorithme mettant en œuvre la méthode.

Préservation des d.f. lors d'une décomposition

- Le processus de décomposition d'une relation consiste à remplacer cette relation par un ensemble de relations (au minimum 2).
- Normalement Une telle décomposition **doit préserver les d.f.** vérifiées par la relation de départ qui sont des propriétés indépendantes du temps et existant entre les attributs.
- On dit que la décomposition d'une relation R vérifiant un ensemble **F** de d.f en {R1, R2, R3,....Rn} préserve les d.f.
  - **si la fermeture F^+ de l'ensemble F** est égale à la fermeture de l'union des d.f. **vérifiées** par **R1, R2 ,.... et Rn.**

$$F^+ = [\ FR1 \cup FR2 \cup FR3 \cup \ldots\ldots\ldots \cup FRn]^+$$

- ou $\forall$ i, FRi = {ensemble des d.f. $\mathbf{V} \longrightarrow \mathbf{U}$ de F^+ tel que les attributs V et U appartiennent au schéma de Ri}

Décomposition binaire d'une relation

- Soit R(Ω) un schéma relationnel avec X, Y, Z une partition de Ω (Ω=X$\cup$ Y$\cup$Z).
- Si $X \longrightarrow Y$ est une d.f. vérifiée par R,

  alors **on peut décomposer R** sans perte d'information en **2 relations** : R1(X,Y) et R2(X,Z) telle que :

$$R(\Omega) = R1(X,Y) \blacktriangleright\blacktriangleleft R1(X,Z) \quad (\blacktriangleright\blacktriangleleft : \text{Join})$$

- La relation R1 est construite sur les attributs intervenants dans la dépendance fonctionnelle $X \longrightarrow Y$ c'est à dire X$\cup$Y qu'on note aussi X,Y.
- La relation R2 est construite sur les attributs Ω - Y c'est à dire X$\cup$Z qu'on note aussi X, Z.

exemple

La relation ADRESSE qui vérifie la d.f. Code_Postal ⟶ Ville (qui l'empêche d'être en BCNF) peut être décomposée sur la base de cette d.f. en deux relations :

- **R1(Code_Postal, Ville)**
  - vérifiant la d.f. {Code_Postal ⟶ Ville }
  - et ayant donc pour clé l'attribut Code_Postal
- **R2(Code_Postal, Nom_Rue)**
  - vérifiant uniquement les d.f. triviales qu'on peut énoncer en utilisant la propriété de réflexivité.
  - R2 a pour clé le groupe d'attributs (Code_Postal, Nom_Rue)
- Et on a ADRESSE =R1 ▶◀ R2

- **La relation** R1 ▶◀ R2

  a le même schéma que ADRESSE et même extension.

  ⇒ cette décomposition préserve bien la jointure (i.e. les données).

- **Cependant, elle ne préserve pas les d.f. car :**

Les d.f. associées à R1 sont : FR1 = {Code_Postal ⟶ Ville }

Les d.f. associées à R2 sont : FR2 = {d.f. triviales} telles que :

(Code_Postal, Nom_Rue) ⟶ Code_Postal ou (Code_Postal ,Nom_Rue) ⟶ Nom_Rue }

d'où : (FR1 ∪ FR2)= {Code_Postal ⟶ Ville }

On voit que (FR1 ∪ FR2)+ ≠ F+

- Par exemple la d.f. (Ville, Nom_Rue) ⟶ Code_Postal qui était vérifiée par la relation de départ ADRESSE est perdue lorsqu'on décompose cette relation afin de la mettre en BCNF).
- Pour vérifier que la d.f. **(Ville, Nom_Rue)** ⟶ **Code_Postal** est perdue, il faudra utiliser l'algorithme de calcul de la fermeture d'un ensemble d'attributs à partir de **(FR1 ∪ FR2).**

Décomposition binaire sans perte d'information

- Soit **R (Ω)** un schéma relationnel vérifiant un ensemble **F** de d.f.
- Une décomposition **(R1, R2)** de **R** **se fait sans perte d'information si au moins** l'une des deux d.f. suivantes appartient à **F +** :

(1) **R1 ∩ R2 ⟶ R1 - R2**

(2) **R1 ∩ R2 ⟶ R2 - R1**

avec :

R1 ∩ R2 = { ensemble des attributs ∈ à R1 et ∈ à R2 aussi)

R1 - R2 = { ensemble des attributs ∈ à R1 et ∉ à R2)

R2 - R1 = { ensemble des attributs ∈ à R2 et ∉ à R1)

exemple :

- Soit la relation R(A, B, C) vérifiant les d.f. F ={ C ⟶A ; B ⟶ C}

  considérons la décomposition de R en { R1(A,C) , R2(B, C) } ,

  on a : R1 ∩ R2 = { C } ; R1 - R2 = { A } et R2 - R1 = { B}

  On s'aperçoit que la dépendance fonctionnelle C ⟶ A

  (c.a.d. R1 ∩ R2 ⟶ R1 - R2) appartient à F et donc nécessairement à F + aussi.

  Cette décomposition **se fait donc sans perte d'informations**.

exemple :

- **Si on considère maintenant la décomposition** de R en { R1(A,C) , R2(A, B) }

  on a : R1 ∩ R2 = { A } ; R1 - R2 = { C } et R2 - R1 = { B}

On s'aperçoit que

ni la d.f. A ⟶ C (c.a.d. R1 ∩ R2 ⟶ R1 - R2)

ni A ⟶ B (c.a.d. R1 ∩ R2 ⟶ R2 - R1)

n'appartient à F ni à F+.

Cette décomposition **se fait donc avec perte d'informations** (ne pas confondre A ⟶ C avec C ⟶ A !).

Décomposition : Les méthodes agrégatives

- basées sur un algorithme ayant comme entrée une **couverture minimale** de l'ensemble F de d.f.
- **Résultat :** un ensemble d**e relations qui sont au moins en 3FN**
- on l'appelle généralement algorithme de décomposition en 3FN.
- S'utilise pour décomposer n'importe quelle relation qui est au plus en 2FN et au moins en 1FN (attributs atomiques) et vérifiant un ensemble F de d.f.
- Mais **nécessite** donc de **trouver** d'abord une **couverture minimale pour l'ensemble F.**

Soit R(Ω) Le principe est le suivant :

⓪ S'il existe dans la couverture des d.f X ⟶ A1 ; ... ; X ⟶ An tel que **X U A1 U....An = Ω** alors retourner comme résultat R elle-même.

① Extraire tous les attributs isolés c'est à dire qui ne participent dans aucune d.f., et construire une relation ayant comme attributs ces attributs isolés et pour **clé** la clé par défaut c'est à dire **l'ensemble de ses attributs** ;

② Regrouper les d.f. de la couverture **par paquets** chacun contenant uniquement les dépendances ayant **la même partie gauche** (au pire on aura une D.F. par paquet !) ;

③ Prendre le paquet ayant **le plus grand nombre** de D.F.

Ces D.F. sont toutes de la forme :

X ⟶ A1 ; X ⟶ A2 ; ..... ; X ⟶ An

car on sait qu'on travaille avec la couverture minimale qui par définition contient uniquement des d.f. ayant **une partie droite** unique (**un seul attribut**)

ce qui n'est pas obligatoire pour la partie gauche qu'on a désigné par X exprès.

④ **Construire une relation** dont la liste des attributs est :

(X , A1 , A2,.....An) et dont le nom sera attribué en fonction des besoins de l'application !.

Cette relation est bien **en 3FN** car

sa seule clé est l'ensemble d'attributs X

et il n'existe pas de D.F. transitive sur cette clé.

En effet si on a deux D.F. X ⟶ Ai et X ⟶ Aj

qui appartiennent à la couverture minimale, on est sûr que la D.F. **Ai ⟶ Aj qui pourrait causer une transitivité ne figure pas dans la couverture.**

En effet, si Ai ⟶ Aj figurait dans la couverture avec X ⟶ Ai on n'aurait pas eu la D.F. X ⟶ Aj dans la couverture minimale puisqu'elle serait redondante et peut s'obtenir alors à partir de : X ⟶ Ai et Ai ⟶Aj en appliquant la propriété de transitivité.

⑤ **Eliminer le paquet** en cours d'examen de la couverture minimale

puis éliminer tous les attributs qui deviennent isolés (qui ne participent dans aucune dépendance fonctionnelle parmi celles qui restent)

⑥ S'il reste des paquets non traités Aller en ③

Exemple d'application de l'algorithme

- Soit la relation **COURS(**PROF,MODULE, SALLE, HEURE**)** vérifiant les d.f. suivantes :

F = { PROF,MODULE ⟶ SALLE ;
MODULE, SALLE ⟶ HEURE ;
PROF,MODULE, HEURE ⟶ MODULE ;
PROF,MODULE, SALLE ⟶ HEURE, SALLE**}**

- La première étape de l'algorithme est de chercher une couverture minimale pour F.

- On doit appliquer la propriété de décomposition aux d.f. dont les parties droites comporte plusieurs attributs pour se conformer à la propriété N°1 de la couverture minimale (**partie droite unique**)

On aura l'ensemble **F1** :

1 PROF,MODULE⟶SALLE

2 MODULE, SALLE⟶HEURE

3 PROF,MODULE, HEURE⟶MODULE

4 PROF,MODULE, SALLE⟶HEURE

5 PROF,MODULE, SALLE⟶SALLE

- **On élimine** ensuite de **F1** les d.f. obtenues par la propriété **d'augmentation** pour se conformer à la **propriété N°2 de la couverture minimale**.

- C'est le cas par exemple des d.f. 4 et 5 qui s'obtiennent par augmentation à partir de 1 et 2 respectivement.

On aura l'ensemble **F2** :

1 PROF,MODULE⟶SALLE

2 MODULE, SALLE⟶HEURE

3 PROF,MODULE, HEURE⟶MODULE

- **On élimine** aussi de **F2** les d.f. triviales obtenues par la propriété de **réflexivité** pour se conformer à la propriété N°2 de la couverture minimale.

  C'est le cas de la d.f. **3** ci-dessus.

➢ On aura l'ensemble **F3** :

1 PROF,MODULE⟶SALLE

2 MODULE, SALLE⟶HEURE

➢ **On élimine** ensuite les **d.f. redondantes** dans **F3** pour se conformer à la propriété N° 3 de la couverture.

➢ On utilise l'algorithme de calcul de la fermeture d'un ensemble d'attributs (**X+**) qui permet de vérifier si une d.f. est redondante dans un ensemble

(PROF,MODULE)**+** = (PROF, MODULE)

(MODULE, SALLE) **+** = (MODULE, SALLE)

On a : **SALLE** $\not\subset$ **(PROF,MODULE)+** donc la **d.f. 1 n'est pas redondante** dans F3.

On a : **HEURE** $\not\subset$ **(MODULE, SALLE)+** donc la **d.f. 2 n'est pas redondante** dans F3.

☞ Ainsi **F3 constitue** une **couverture minimale pour F.**

- L'application de l'algorithme de décomposition en 3FN consiste donc à regrouper les d.f. de F3 en paquets chacun contenant toutes les d.f. ayant la même partie gauche.
- Pour notre exemple, on aura une d.f. par paquet et qui sont :

  P1 : PROF,MODULE ⟶ SALLE

  P2 : MODULE, SALLE ⟶ HEURE

  On initialise une variable **LISTE_ATTRIBUTS={ PROF,MODULE, SALLE, HEURE}** avec l'ensemble des attributs de la relation.

  Il n'existe pas d'attributs isolés (ne participant à aucune d.f), donc aucune relation n'est à construire pour ce type d'attributs (étape 0)

- On construite une relation **P1(**PROF,MODULE, SALLE) avec le paquet P1.

On supprime la d.f. 1 de F3. On aura :

F'3 = { MODULE, SALLE ⟶ HEURE}.

On supprime de LISTE_ATTRIBUTS les attributs qui sont devenus isolés dans F'3.

C'est le cas de l'attribut **PROF** qui ne participe dans aucune d.f. de F'3.

On aura LISTE_ATTRIBUTS={MODULE, SALLE, HEURE}.

- On construite une autre relation **P2(**MODULE, SALLE, HEURE) avec le paquet P2. On supprime la d.f. 2 de F'3. On aura :

F''3 = { ∅ }

On supprime de LISTE_ATTRIBUTS les attributs qui sont devenus isolés dans F''3. Tous les attributs sont isolés car F''3 est vide.

On aura LISTE_ATTRIBUTS={∅}.

- On a examiné tous les paquets. On arrête l'algorithme avec comme résultat la décomposition suivante :

  P1(PROF,MODULE, SALLE)

  vérifiant la d.f. PROF, MODULE⟶ SALLE

  et ayant pour clé le couple (PROF, MODULE).

  P1 est au moins en 3FN.

  P2(MODULE, SALLE, HEURE)

  vérifiant la d.f. MODULE, SALLE ⟶ HEURE

  et ayant pour clé le couple (MODULE, SALLE)

  P2 est au moins en 3FN.

- Cette décomposition préserve les d.f. car **FP1 ∪ FP2 = F3**.

- Et puisque **F3 est une couverture minimale**, on sait par définition **qu'elle est équivalente à F.**

Remarque

- **Les méthodes dites agrégatives** se caractérisent par le fait qu'elles donnent comme résultat une décomposition qui préserve les d.f. **mais pas nécessairement les données.**

☝ Il a été montré qu'étant donnée une décomposition d'une relation R selon cette méthode et qui ne préserve pas les données

- **il suffit de lui rajouter une nouvelle relation S(X) dans laquelle X est une clé de R pour constituer une décomposition préservant en même temps les dépendances fonctionnelles et les données.**

Les méthodes par décomposition (ou descendantes)

- basées sur un processus itératif consistant à décomposer à chaque étape une relation par application du théorème de décomposition binaire présenté plus haut.
- A la fin du processus, on obtient un arbre de décomposition dans lequel toutes les feuilles sont les relations constituant la décomposition.
- Le théorème de décomposition binaire est applicable quelque soit la relation à décomposer et quelle que soit sa forme normale.
- C'est donc au concepteur de choisir la bonne dépendance fonctionnelle à isoler grâce à ce théorème.
- Les situations possibles peuvent être résumées comme suit :

Principe de la méthode de décomposition descendante

Type de la Relation	Type de d.f. à utiliser	Résultat
Relation en **1FN** et **non en 2FN**	une d.f. qui contredit la 2FN	2 relations dont une est au moins en 2FN
Relation en **2FN** mais **non en 3FN**	une d.f. qui contredit la 3FN	2 relations dont une est au moins en 3FN
Relation en **3FN** mais **non en BCNF**	une d.f. qui contredit la BCNF	2 relations dont une est au moins en BCFN

- Il faut remarquer aussi qu'il peut exister pour une même relation plusieurs d.f. qui contredisent telle ou telle forme normale.
- C'est pour cela que le choix d'une d.f. à utiliser avec le théorème de décomposition binaire conduit à une décomposition qui serait différente que celle qu'on obtiendrait avec une autre d.f.

  on sait qu'une même relation peut être décomposée de plusieurs façons différentes.

- **Ce qui qualifie** les décompositions faites sur la base du théorème de décomposition binaire, **c'est qu'elles se font sans perte d'informations**.

- ⇒ A chaque application du théorème si l'une des deux relations résultantes n'est pas dans la forme normale que l'on souhaite, c'est qu'elle vérifie une d.f. qui l'empêche d'être dans cette forme là.

  - **Il faudra alors réappliquer le théorème à cette relation sur la base de cette d.f.**

- On voit que c'est effectivement un processus itératif qui se termine lorsque toutes les relations de l'arborescence sont dans une forme normale souhaitée par le concepteur.

Exemple :

Soit la relation **R(NUMERO, NOM, SPECIALITE, VILLE)**

vérifiant les d.f. :

NUMERO ⟶ **NOM** (1)

VILLE ⟶ **SPECIALITE** (2)

On peut facilement montrer que le couple **(NUMERO,VILLE) est une clé** pour la relation.

Grâce a la réflexivité, on peut énoncer que :

NUMERO,VILLE ⟶ **NUMERO** (3) **NUMERO,VILLE** ⟶ **VILLE** (4)

Par application de la transitivité à (3) et (1) on déduit que :

NUMERO,VILLE ⟶ **NOM** (5).

Par application de la transitivité à (4) et (2) on déduit que :

NUMERO,VILLE ⟶ **SPECIALITE (**6)

- Puisque le couple d'attributs **(NUMERO,VILLE)** détermine tous les autres attributs de la relation, il constitue bien une clé pour la relation.

Exemple :

Cette relation est simplement en 1FN et n'est pas en 2FN car les deux D.F.

NUMERO ⟶ **NOM** et **VILLE** ⟶ **SPECIALITE**

sont des dépendances ayant en partie droite des attributs non primaires et dans la partie gauche une partie de la clé **(NUMERO,VILLE).**

Ceci revient à dire aussi que les d.f. :

(NUMERO,VILLE) ⟶ **NOM et (NUMERO,VILLE)** ⟶ **SPECIALITE**

ne sont pas élémentaires

ou que **NOM ne dépend pas pleinement** de la clef, de même que **SPECIALITE**.

On peut dans ce cas appliquer le théorème de décomposition binaire sur la base de l'une des deux d.f.

NUMERO ⟶ **NOM** ou **VILLE** ⟶ **SPECIALITE**

car elles empêchent toutes les deux la relation d'être en 2FN.

❖ Si on choisit de décomposer selon **VILLE** ⟶ **SPECIALITE**, on aura :

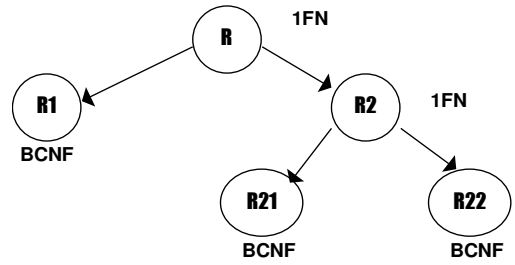
Relation	clé(s)	dépendances	F.N.
R1(VILLE,SPECIALITE)	VILLE	VILLE⟶ SPECIALITE	BCNF
R2(NUMERO,VILLE,NOM)	(NUMERO,VILLE)	NUMERO⟶NOM	1FN

- Puisque **R2** n'est pas en **2FN** à cause de la d.f. **NUMERO** ⟶ **NOM** , on poursuit le processus de décomposition avec **R2**.
- On applique le théorème de décomposition binaire sur la base de la d.f. **NUMERO** ⟶ **NOM** car elle **empêche la relation R2 d'être en 2FN.** on aura :

Relation	clé(s)	Dépendances	F.N.
R21(NUMERO,NOM)	NUMERO	NUMERO⟶ NOM	BCNF
R22(NUMERO,VILLE)	(NUMERO,VILLE)	d.f. triviales	BCNF

➢ Toutes les relations sont maintenant en BCNF, on arrête donc le processus de décomposition.

- Une représentation graphique de l'arbre de décomposition où toutes les feuilles sont des relations en BCNF serait :



- Le résultat du processus de décomposition de la relation R selon la méthode dite par décomposition sera donc constitué des trois relations R1, R21 et R22 qui constituent des feuilles de l'arbre ci-dessus.

Remarque

- Les méthodes dites par décomposition se caractérisent par :
  - donnent comme résultat une décomposition qui **préserve les données**
  - **Mais pas nécessairement** les **d.f.**

❶ Algèbre Relationnelle

Les LM de données relationnelles

- **Un LMD se compose de :**

  - un ensemble de commandes permettant **d'interroger** une BD
  - un ensemble de commandes permettant de la **modifier** (insertion, suppression, mise a jour, etc.).

➢ En général, la plupart des SGBD propose a l'utilisateur un LMD avec une interface conversationnelle et aussi une possibilité d'intégration de ce LMD dans un langage hôte.

- De manière générale, L'algèbre relationnelle consiste à munir le domaine des relations avec **un ensemble d'opérateurs.**

- **Cet ensemble contient :**

  - **Des opérateurs de base**

  - **+ des opérateurs complémentaires qui peuvent s'exprimer en fonction des opérateurs de base** mais qui ont une utilité pratique

• Les opérations de base

Il existe 2 catégories :

1. Les opérations **binaires** **s'appliquant à 2 relations**
2. Les opérations **unaires** **s'appliquant à 1 seule relation**

Panorama des opérations binaires

Panorama des opérations binaires

L'union de deux relations R et S

- **R et S ont la même arité i.e. le même nombre de colonnes**
- **Notation : R ∪ S ou UNION(R,S)**
- **Résultat : une relation contenant l'ensemble des tuples appartenant a R ou à S ou au deux.**

Remarques

- Les relations R et S **doivent normalement avoir le même schéma** (c.a.d. les mêmes attributs) qui sera dans ce cas le schéma de la relation résultat.
- Si les deux relations n'ont pas la même sémantique **l'union des deux n'a sans doute pas de sens** mais elle peut être calculée.

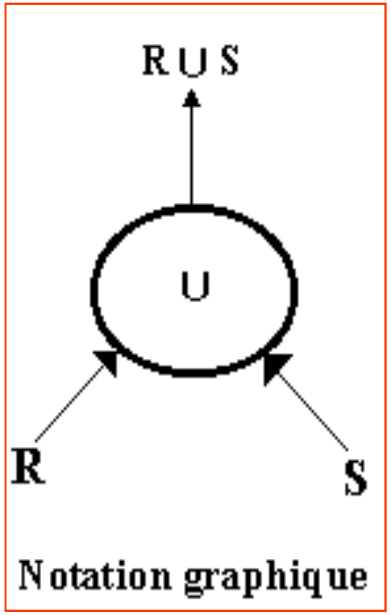
Exemple : Union entre deux relations R et S ayant même schéma

A	B	C
a1	b1	c1
a2	b2	c2
a3	b3	c3

Relation R

A	B	C
a1	b1	c1
a4	b4	c4
a5	b5	c5
a2	b2	c2

Relation S



Notation graphique

A	B	C
a1	b1	c1
a2	b2	c2
a3	b3	c3
a4	b4	c4
a5	b5	c5

Relation R ∪ S

La différence entre deux relations R et S

- **R et S ont la même arité i.e. le même nombre de colonnes**
- **Notation : R — S ou MINUS(R,S)**
- **Résultat : une relation contenant l'ensemble des tuples appartenant à R et n'appartenant pas à S.**

Remarques

- Les relations R et S **doivent normalement avoir le même schéma** (c.a.d. les mêmes attributs) qui sera dans ce cas le schéma de la relation résultat.
- Si les deux relations n'ont pas la même sémantique **la différence n'a sans doute pas de sens** même si on peut la calculer.

Pb : ? Schéma de la relation résultat = celui de R ou de celui de S ou Autre

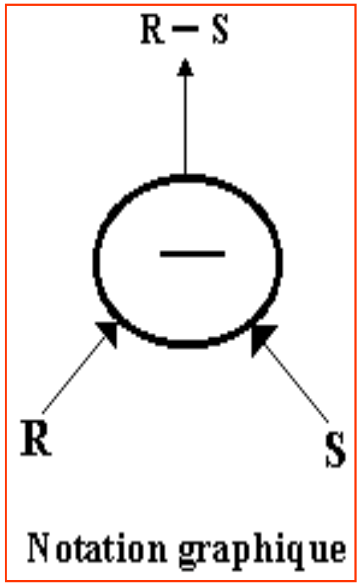
Exemple : Différence entre deux relations R et S ayant même schéma

A	B	C
a1	b1	c1
a2	b2	c2
a3	b3	c3

Relation R

A	B	C
a1	b1	c1
a4	b4	c4
a5	b5	c5
a2	b2	c2

Relation S



Notation graphique

A	B	C
a3	b3	c3

Relation R-S

Résultat

- Un seul tuple **(a3,b3,c3)** appartient à R et n'appartient pas à S.
- les tuples (a1,b1,c1) et (a2,b2,c2) appartiennent à R mais aussi à S. Donc ils ne doivent pas figurer dans la relation résultat R - S.

La différence n'est pas commutative

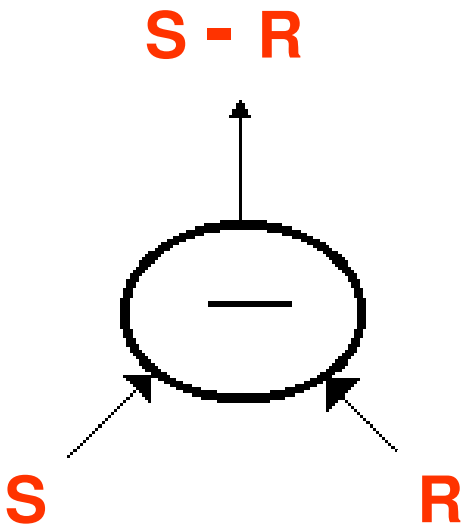
$$R - S \neq S - R$$

A	B	C
a1	b1	c1
a2	b2	c2
a3	b3	c3

Relation R

A	B	C
a1	b1	c1
a4	b4	c4
a5	b5	c5
a2	b2	c2

Relation S



Notation graphique

?

Résultat

- **Produit Cartésien de deux relations R et S**

  - R et S ont des schémas quelconques
  - Notation : R X S ou PRODUCT (R,S)
  - **Résultat :**

    - une relation ayant pour attributs la concaténation des attributs de R et de S
    - et contenant tous les tuples obtenus par concaténation d'un tuple de R a un tuple de S.

☝ **Le calcul du produit cartésien impose que les attributs de R et de S n'aient pas de nom commun même s'ils ont le même domaine.**

Exemple : Produit cartésien entre 2 relations R et S

- Le produit cartésien des relations **R** et **S** utilisées dans l'exemple de l'union ***ne peut se faire que si on donne aux attributs de S (A,B et C) des noms qui seront distincts de ceux de R*** (car R a aussi des attributs ayant les mêmes noms A, B et C).

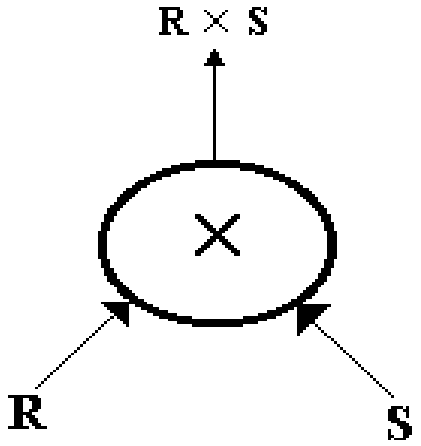
- Si on renomme dans S la colonne A par **D**, la colonne B par **E** et la colonne C par **F**, on aura :

A	B	C
a1	b1	c1
a2	b2	c2
a3	b3	c3

Relation R

D	E	F
a1	b1	c1
a4	b4	c4
a5	b5	c5
a2	b2	c2

Relation S



Notation graphique

- Le résultat sera une nouvelle relation ayant pour schéma et pour extension :

A	B	C	D	E	F
a1	b1	c1	a1	b1	c1
a1	b1	c1	a4	b4	c4
a1	b1	c1	a5	b5	c5
a1	b1	c1	a2	b2	c2
a2	b2	c2	a1	b1	c1
a2	b2	c2	a4	b4	c4
a2	b2	c2	a5	b5	c5
a2	b2	c2	a2	b2	c2
a3	b3	c3	a1	b1	c1
a3	b3	c3	a4	b4	c4
a3	b3	c3	a5	b5	c5
a3	b3	c3	a2	b2	c2

Relation R x S

- Si le nombre de tuples de R est **n** et le nombre de tuples de S est **m** alors la relation R x S contiendra $n*m$ tuples

- dans l'exemple R x S contient 12 tuples puisque R contient 3 tuples et S en contient 4.

☝ Le Produit cartésien **est-il commutatif ou non ?**

$$R \times S \neq S \times R$$

ou $R \times S = S \times R$

Panorama des opérations unaires

- Il existe **2** opérations unaires appelées **PROJECTION** et **SELECTION** (ou **RESTRICTION**).

- **Ces opérations consistent respectivement à :**

  - Eliminer des colonnes (**PROJECTION**) dans une relation

ou

  - Eliminer de lignes (**SELECTION**) dans une relation.

✍ La combinaison de ces deux opérations avec les opérations binaires permet de générer toutes les opérations de l'algèbre relationnelle.

- **Projection d'une relation ayant pour schéma R(A1,A2,....An)**

  - **Notation : Π Ai1,Ai2,....Aip (R) ou PROEJECT Ai1,Ai2,....Aip (R)**

  - **Résultat :**

    - une relation R' ayant pour schéma R'(Ai1,Ai2,....Aip)

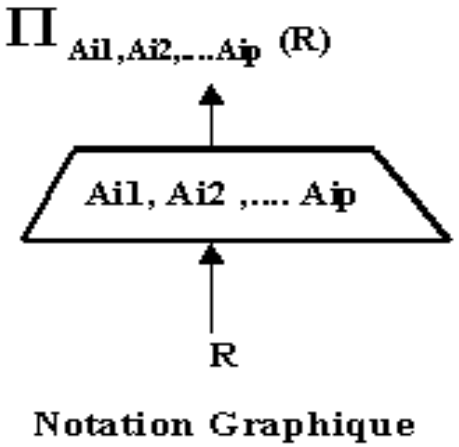
    - **Contenant les tuples obtenus par élimination des valeurs des attributs de R n'appartenant pas a R' (c.a.d. à Ai1,Ai2,....Aip) et par suppression des tuples en double de la relation résultat (c.a.d. de R').**

Exemple : Projection R' = Π A , C (R)

- **R' = Π A , C (R)** aura pour schéma **R'(A,C)**
- On élimine les valeurs des attributs B (la colonne B de la relation R) car l'attribut B n'appartient pas a R')
- et pour extension

A	B	C
a1	b1	c1
a2	b2	c2
a3	b3	c3

Relation R



Notation Graphique

A	C
a1	c1
a2	c2
a3	c3

résultat : R' (A,C)

- **Sélection ou Restriction dans une relation ayant pour schéma R(A1,A2,….An)**

  - Consiste à éliminer tous les tuples (les lignes) dont les attributs ne vérifient pas une condition ***c*** spécifiée.

  - **Notation : σ_c ou SELECT c (R)**

  - **Résultat :**

    - **une relation R' de même schéma que R**
    - Contenant tous les tuples de R **qui satisfont la condition *c*.**

- **La condition *C* spécifiée**

  est une expression faisant intervenir :

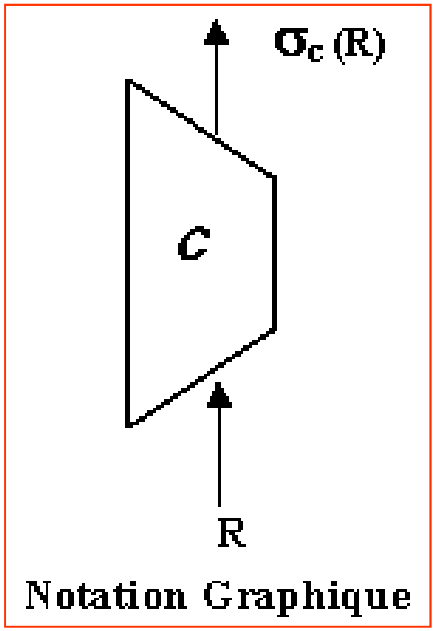
  - **des opérandes (constantes ou des noms d'attributs)**
  - **des opérateurs arithmétiques de comparaison (≤ , ≠ , < , = , > , ≥)**
  - **des opérateurs logiques (ET,OU,NON)**

Exemple : R'= SELECT "prix ≤ 500 ET qte ≥ 20" (PRODUIT)

- La relation **PRODUIT** a pour schéma **PRODUIT (NOM_PRODUIT,PRIX,QTE)**
- La condition ***C*** est donc : **"prix ≤ 500 ET qte ≥ 20"**

NOM_PRODUIT	PRIX	QTE
BOULONS	10	100
VIS	15	50
CHAISE	500	10
TABLE	1000	20

Extension de la relation PRODUIT



Notation Graphique

Résultat

NOM_PRODUIT	PRIX	QTE
VIS	15	50
BOULONS	10	100

σ_c (PRODUIT)

Remarque

- Si on avait comme condition ***C : "prix < 10 ET qte ≥ 20"***
- le résultat serait une relation vide (ne contenant aucun tuple) car aucun tuple de la relation PRODUIT ne vérifie cette condition.

NOM_PRODUIT	PRIX	QTE

Résultat : une relation vide

Opérations complémentaires

- **Les opérations de base** permettent d'exprimer toutes les questions qu'on veut poser à une BDR. **ELLES forment un ensemble cohérent et minimal.**

- **Dans la pratique on dispose aussi d'opérations dites complémentaires qui ne modifient pas l'algèbre ⇒ on peut les obtenir a partir des opérations de base**

  - **MAIS ELLES sont très utiles pour la réalisation pratique d'un SGBD relationnel.**

Par Exemple :

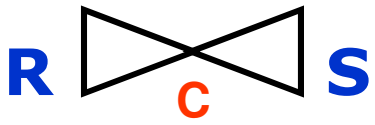
- Le produit cartésien est très coûteux en temps machine et en espace ⇒ **On préfère réaliser une autre opération similaire mais plus restrictive qu'on appelle la jointure.**

La Jointure entre deux relations R et S

La Jointure entre deux relations R et S

- *On distingue différents types de JOINTURE*

1. La jointure selon une condition (ou une qualification)

- **Une condition C comparant des attributs de R à ceux de S**
- **Notation : JOIN $_C$ (R,S) ou** 

- **Résultat :**

une relation de schéma égal a **l'union des schémas** de R et de S
contenant l'ensemble des tuples du produit cartésien **R x S**
satisfaisant la qualification **C**.

Exemple : Jointure selon une condition entre 2 relations R et S

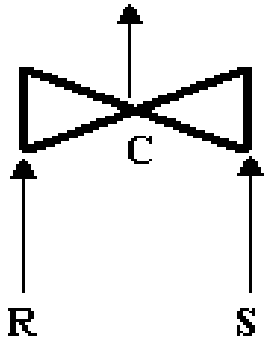
Le résultat de la jointure de R et de S selon la condition **c** : **" C ≤ D"**
Donnera pour Résultat : une relation **R'** de schéma **R'(A,B,C,D,E)**

A	B	C
1	2	3
4	5	6
7	8	9

Relation R

D	E
4	1
6	2

Relation S



Notation graphique

Relation R'

A	B	C	D	E
1	2	3	4	1
1	2	3	6	2
4	5	6	6	2

- La forme la plus courante de la qualification **C** est : **A** θ **B** ou A est un attribut de R et B un attribut de S et θ un opérateur de comparaison ($\leq$, $\neq$, $<$, $=$, $>$, $\geq$).

- Dans le cas ou l'opérateur θ est l'opérateur d'égalité (=) la jointure s'appelle une **EQUIJOINTURE**.

- Dans le cas contraire on dit aussi une **θ-JOINTURE**.

La jointure étant une opération complémentaire :

Elle peut être exprimée à l'aide des opérations de base :

SELECTION et PRODUIT CARTESIEN

- **En effet on a :**

$$\text{JOIN}_{c}(R,S) = \text{SELECT}_{c}(R \times S)$$

Ce qui revient à :

- **Calculer le produit cartésien R x S**
- **Appliquer une opération de sélection selon la condition c à la relation résultant du produit cartésien**

La JOINTURE Naturelle entre deux relations R et S

- **R et S ont des schémas non disjoints** (R et S ont des attributs de même noms)

  - **Notation : JOIN (R,S) ou** $R \bowtie S$

  - **Résultat :**

    - une relation de schéma égal a l'union des schémas de R et de S
    - contenant tous les tuples obtenus par EQUIJOINTURE de R et de S sur tous les attributs ayant le même nom dans R et S suivie d'une PROJECTION qui permet de conserver un seul de ces attributs égaux de même nom.

Exemple : Jointure Naturelle entre 2 relations R et S

En supposant que **A1,A2,.....Ak** sont les noms d'attributs apparaissant dans les deux relations R et S, **la condition c** qui sera utilisée dans **l'équijointure** sera de la forme :

$$R.A1=S.A1 \wedge R.A2=S.A2 \wedge \ldots\ldots \wedge R.Ak=S.Ak$$

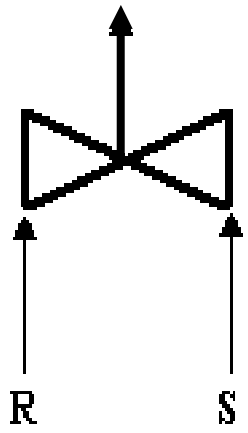
❖ **Soient les 2 relations suivantes**

A	B	C
a1	b1	c1
a2	b1	c1
a3	b3	c3

Relation R

B	C	D
b1	c1	d1
b1	c1	d2
b3	c3	d3

Relation S



Notation graphique

La jointure naturelle de R et S sera une relation **R'** ayant pour schéma :

R' [(A,B,C) ∪ (B,C,D)] c.a.d. **R'(A,B,C,D)**

Les attributs de même noms dans R et S sont B et C.

Donc **la condition c** pour réaliser **l'equijointure** sera :

$$R.B = S.B \wedge R.C = S.C .$$

Le résultat de la jointure naturelle de R et S

Relation R' = JOIN (R,S)

A	B	C	D
a1	b1	c1	d1
a1	b1	c1	d2
a2	b1	c1	d1
a2	b1	c1	d2
a3	b3	c3	d3

La Jointure Naturelle étant une opération complémentaire, Elle peut être exprimée à l'aide des opérations de base :

SELECTION , PROJECTION et PRODUIT CARTESIEN

- **En effet on a :**

- $JOIN\ (R,S) = PROJECT_{v}\ (SELECT_{c}\ (R \times S))$

  ou **v** est égal l'union des attributs de R et des attributs de S **n'appartenant pas à R**.

- Le calcul de la jointure naturelle passe donc par les étapes intermédiaires suivantes :

1. **Calcul de la relation $R1 = R \times S$**
2. **Calcul de la relation $R2 = SELECT_{C}\ (R1)$ avec la condition c :**

$$(R.B = S.B \wedge R.C = S.C)$$

3. **Calcul de la relation $R3 = \Pi_{A, B, C, D}\ (R2)$ qui sera le résultat final**

$$\Rightarrow JOIN\ (R,S) = \Pi_{A, B, C, D}\ (SELECT_{C}\ (R \times S))$$

La SEMI-JOINTURE entre deux relations R et S

- C'est une opération dérivée de la jointure.
- Consiste à faire une jointure entre R et S puis de projeter la relation obtenue sur les attributs de la relation R (on ne garde que les attributs de R)
- R et S ont des schémas quelconques (peuvent être disjoints)

➢ Notation : $JOIN_c(R,S)$ ou $R \ltimes_c S$

❖ Résultat :

- une relation de même schéma que R
- contenant l'ensemble des tuples de R participant à la jointure de R et de S selon la condition c.

Exemple : Semi-Jointure entre 2 relations R et S

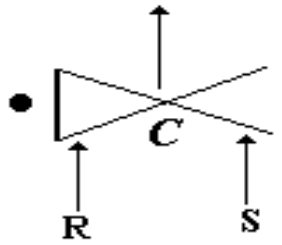
- La semi-jointure de R et S selon la condition **C** : **"C ≤ D "**

A	B	C
1	2	3
4	5	6
7	8	9

Relation R

D	E
4	1
6	2

Relation S



Notation graphique

- **1) une jointure de R et S selon la condition C**
- **2) une projection de la relation résultant de cette jointure sur les attributs (A,B,C) qui sont les attributs de R**

Ce qui donnera :

1) La jointure de R et S selon la condition C : "C ≤ D " donnera :

A	B	C	D	E
1	2	3	4	1
1	2	3	6	2
4	5	6	6	2

2) la projection du résultat sur les attributs (A,B,C) qui donnera come résultat final

A	B	C
1	2	3
4	5	6

$\Pi_{A,B,C}$ (JOIN "C ≤ D" (R,S))

L'intersection entre deux relations R et S

Intersection entre deux relations R et S

L'**INTERSECTION** de deux relations R et S **de même schéma est une relation** :

➢

- **ayant même schéma que R et S**
- **contenant les tuples qui appartiennent à R et à S en même temps.**

➢ **Notation : R ∩ S ou ou INTERSECT(R,S)**

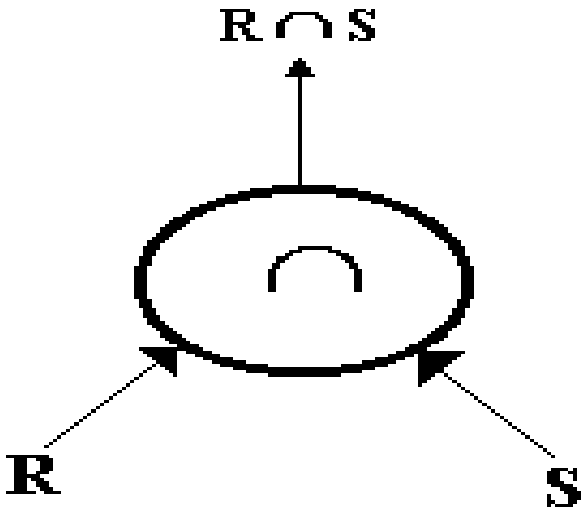
Exemple : L'intersection entre les relations R et S

A	B	C
a1	b1	c1
a2	b3	c2
a2	b2	c1
a3	b3	c3

Relation R

A	B	C
a1	b1	c1
a2	b2	c2
a2	b2	c1

Relation S



Notation graphique

A	B	C
a1	b1	c1
a2	b2	c1

Relation R ∩ S

Résultat

L'opération INTERSECTION étant complémentaire peut s'obtenir a partir de l'opération de base DIFFERENCE à l'aide de l'une des deux formules suivantes :

a) $R \cap S = R - (R - S)$

b) $R \cap S = S - (S - R)$

En effet si on pose : $R = R_{+S} \cup R_{-S}$ **où**

- R_{+S} représente l'ensemble des tuples de **R appartenant aussi à S**

et

- R_{-S} représente l'ensemble des tuples de **R n'appartenant pas à S.**

Et si on pose : $S = S_{+R} \cup S_{-R}$ **où**

- S_{+R} représente l'ensemble des tuples de S **appartenant aussi à R**

et

- S_{-R} représente l'ensemble des tuples de S **n'appartenant pas à R.**

☞ **on remarque que :**

Les deux ensembles S_{+R} **et** R_{+S} **sont égaux**

Par définition de l'opération **DIFFERENCE** on a aussi :

- **R - S = R_{-S}** (l'ensemble des tuples de R n'appartenant pas à S)

et

- **S - R = S_{-R}** (l'ensemble des tuples de S n'appartenant pas à R)

On peut alors écrire d'après a) $R \cap S = R - (R - S)$

que :

$$R \cap S = R_{+s} \cup R_{-s} - (R_{-s}) = R_{+s}$$

Et on sait que R_{+s} représente l'ensemble des tuples de **R** appartenant aussi à **S**

C'est donc bien la définition de l'opération $R \cap S$

On peut écrire aussi d'après b) $R \cap S = S - (S - R)$

que :

$$R \cap S = S_{+R} \cup S_{-R} - (S_{-R}) = S_{+R}$$

Et sachant que S_{+R} représente l'ensemble des tuples de S appartenant aussi à R

Ceci n'est autre que la définition de l'opération $R \cap S$

La DIVISION d'une relation R par une relation S

- **Ou Quotient**
- **R de schéma R(A1,A2,....An)**
- **S de schéma S(Ap+1,Ap+2,....An)** c.a.d un sous-schéma de R
- **Notation : R % S ou DIVISION(R,S)**
- **Résultat :**
  - une relation **Q** de schéma Q(A1,A2,....Ap)
  - contenant tous les p_tuples de R qui concaténés à chacun des tuples de S donnent toujours un tuple de R.

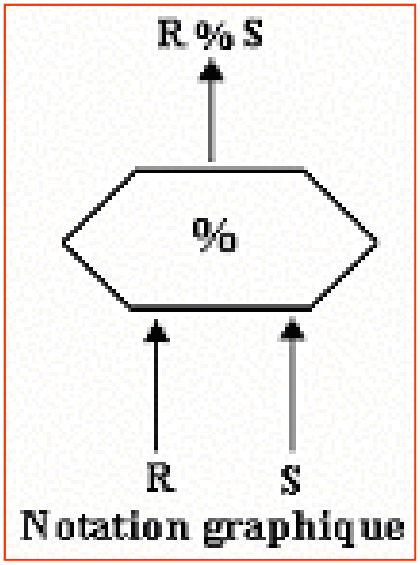
Exemple : DIVISION d'une relations R par S

A	B	C	D
a1	b1	c1	d1
a1	b1	c2	d2
a3	b3	c1	d1
a3	b3	c2	d2
a1	b1	c3	d3

Relation R

C	D
c1	d1
c2	d2

Relation S



Notation graphique

A	B
a1	b1
a3	b3

Relation Q = R % S

- **Q = R % S** aura pour schéma **Q(A,B)** **(les attributs de R moins les attributs de S).**
- Elles contiendra les sous-tuples de R répondant à la définition

En effet la concaténation du sous-tuple (a1, b1) de **R** avec chacun des tuples de **S** donne les deux tuples suivants :

(a1, b1, c1, d1) et (a1,b1, c2, d2)

qui appartiennent bien a R.

⇒ **(a1, b1)** appartiendra à **Q**.

De même la concaténation du sous-tuple (a3, b3) de **R** avec les tuples de **S** donne les deux tuples suivants :

(a3, b3, c1, d1) et (a3, b3, c2, d2)

qui appartiennent aussi a R

⇒ **(a3, b3)** appartiendra à **Q**.

☝ Par contre la concaténation du sous-tuple (a2, b2) de **R** avec chacun des tuples de **S** donne les deux tuples suivants :

(a2, b2, c1, d1) **qui appartient à R**

et

(a2, b2, c2, d2) **qui n'appartient pas à R**

⇒ ce qui contredit la définition du quotient qui énonce que :

Tous les tuples obtenus par CONCATENATION DOIVENT APPARTENIR A R ce qui n'est pas le cas pour (a2, b2)

⇒ **(a2, b2) n'appartiendra pas à Q.**

Le quotient étant une opération complémentaire, il peut s'exprimer à partir des opérations de DIFFERENCE, PRODUIT CARTESIEN et de PROJECTION.

Si on pose **v=(A1, A2, ...Ap)** c.a.d. l'ensemble des attributs de **Q**,

on peut vérifier que :

$$Q = \text{PROJECT}_{v}(R) - \text{PROJECT}_{v}((\text{PROJECT}_{v}(R) \times S) - R)$$

5:Π

1:Π

2: x

3:-

4: Π

6:-

Et enfin :

On n'a pas : **Q x S = R**

mais simplement **Q x S ⊂ R**

Propriétés des opérateurs algébriques

- **Les opérations algébriques ont des propriétés entre elles qui permettent de transformer les expressions en changeant l'ordre d'évaluation de ces opérations**

  - **Objectif :**

    - obtenir des expressions équivalentes mais plus efficaces en termes de temps d'évaluation et d'espace mémoire

⇒ **Ces propriétés vont servir essentiellement à l'optimisation des requêtes par le SGBD**

P1- La ***Commutativité et l'Associativité*** sont les propriétés remarquables vérifiées par **la jointure et le produit cartésien :**

- **Produit cartésien**

R1 **x** R2 = R2 **x** R1 **(commutativité)**

R1 **x** (R2 **x** R3) = (R1 **x** R2) **x** R3 **(Associativité)**

- **Jointure**

JOIN (R1 , R2) = **JOIN** (R2 , R1) **(commutativité)**

JOIN (R1 , **JOIN** (R2 , R3)) = **JOIN** (**JOIN** (R1 , R2) , R3) **(Associativité)**

P2- Remplacement d'une cascade de projections

Si **X** (un ensemble d'attributs) est inclus dans **y** (un autre ensemble d'attributs) alors on peut remplacer

Une cascade de projection de la forme :

$$\Pi\ x\ (\Pi\ y\ (R))$$

Par une seule projection : $\Pi_{x}\ (R)$

P3- Remplacement d'une cascade de SELECTIONS

Si on considère le cas d'une sélection simple mono-attribut avec θ un opérateur de comparaison

$\{ <, \leq, >, \geq, =, \neq, \}$, on peut remplacer

Une cascade de SELECTIONS de la forme :

$$\sigma_{"B\ \theta\ d"}(\sigma_{"A\ \theta\ c"}(R))$$

Par une seule opération de sélection :

$$\sigma_{"B\ \theta\ d \wedge A\ \theta\ c"}(R)$$

Ce résultat peut être généralisé à plusieurs attributs.

P4- Commutation d'une SELECTION et d'une PROJECTION

- **Dans le cas ou l'attribut B est inclus dans Y (un ensemble d'attributs)**
- **On peut remplacer** $\sigma_{"B\ \theta\ c"}(\Pi_{Y}(R))$
- ➢
- ➢**par :** $\Pi_{Y}(\sigma_{"B\ \theta\ c"}(R))$

On applique d'abord la SELECTION puis la PROJECTION.

• **Dans le cas ou l'attribut B n'est pas inclus dans Y (un ensemble d'attributs)**

➢ **on peut remplacer :** $\Pi_{Y}(\sigma_{"B\ \theta\ c"}(R))$

par

$$\Pi_{Y}(\underbrace{\sigma_{"B\ \theta\ c"}(\underbrace{\Pi_{Y \cup B}(R)}_{1})}_{2})_{3}$$

P5- Commutation d'une SELECTION et d'une UNION

- $\sigma_{"A\ \theta\ c"}\ (R1 \cup R2) = \sigma_{"A\ \theta\ c"}\ (R1) \cup \sigma_{"A\ \theta\ c"}\ (R2)$

On applique d'abord la SELECTION puis l'UNION

P6- Commutation d'une SELECTION et d'une DIFFERENCE

- $\sigma_{"A\ \theta\ c"}\ (R1 - R2) = \sigma_{"A\ \theta\ c"}\ (R1) - \sigma_{"A\ \theta\ c"}\ (R2)$

On applique d'abord la SELECTION puis la **DIFFERENCE**

P7- Commutation d'une SELECTION et d'un PRODUIT CARTESIEN

Soient deux relations R1(X) et R2(Y) :

Si le critère de sélection est de la forme

$$\text{"A } \theta \text{ c" avec A} \in \text{X}$$

On peut remplacer :

$$\sigma_{\text{"A } \theta \text{ c"}}(\text{R1} \times \text{R2})$$

Par :

$$(\sigma_{\text{"A } \theta \text{ c"}}(\text{R1})) \ \text{X} \ \text{R2}$$

P7b-Commutation d'une SELECTION et d'un PRODUIT CARTESIEN

➢Si le critère de sélection est de la forme

$$\text{"A } \theta \text{ c"} \wedge \text{"B } \theta \text{ d"} \quad \text{avec A} \in \text{X et B} \in \text{Y.}$$

On peut remplacer :

$$\sigma_{\text{"A } \theta \text{ c"} \wedge \text{"B } \theta \text{ d"}} (R1 \times R2)$$

Par :

$$(\sigma_{\text{"A } \theta \text{ c"}} (R1)) \times (\sigma_{\text{"B } \theta \text{ d"}} (R2))$$

i.e. On applique d'abord la SELECTION puis le produit cartesien

P8-Commutation d'une PROJECTION et d'un PRODUIT CARTESIEN

Soient deux relations R1(X) et R2(Y)

et A , B et C des ensembles d'attributs tel que :

$A = B \cup C$ avec $B \in X$ et $C \in Y$.

On peut remplacer :

$$\Pi_A (R1 \times R2)$$

Par :

$$\Pi_B (R1) \times \Pi_C (R2)$$

i.e. On applique d'abord la **PROJECTION** puis le **produit cartesien**

P9-Commutation d'une PROJECTION et d'une UNION

Soient deux relations R1(X) et R2(Y)

et A , B et C des ensembles d'attributs tel que :

$A = B \cup C$ avec $B \in X$ et $C \in Y$.

On peut remplacer :

$$\Pi_A (R1 \cup R2)$$

Par :

$$\Pi_B (R1) \cup \Pi_C (R2)$$

i.e. On applique d'abord la **PROJECTION** puis **l'UNION**

Récapitulons l'Utilité de ces propriétés :

- Les propriétés **P2 et P3** ont pour intérêt essentiel de regrouper des opérations (SELECTION ou PROJECTION) en une seule opération.

Les propriétés **P4 , P5 , P6 et P7** montrent qu'on peut faire passer une SELECTION avant :

- une PROJECTION ,
- une UNION,
- une DIFFERENCE
- ou un PRODUIT CARTESIEN.

- L'intérêt d'une telle transformation est d'appliquer la SELECTION le plus tôt possible afin de minimiser le nombre de tuples des relations auxquelles seront appliquées les autres opérations (UNION, PRODUIT CARTESIEN, DIFFERENCE, PROJECTION)

Récapitulons l'Utilité de ces propriétés :

Les propriétés **P8 et P9** montrent qu'on peut faire passer une PROJECTION avant :

- une UNION
- ou un PRODUIT CARTESIEN .

L'intérêt d'une telle transformation est d'appliquer la PROJECTION le plus tôt possible afin de **minimiser le nombre de colonnes** des relations auxquelles seront appliquées l'UNION ou le PRODUIT CARTESIEN.

Expressions de l'algèbre relationnelle

- Les opérations algébriques peuvent être combinées pour former des expressions de l'algèbre relationnelle.

- Il sera donc possible de composer la plupart des questions que l'on peut poser à une base de données relationnelle.

- ces questions peuvent être exprimées à l'aide de successions d'opérations (UNION , DIFFERENCE, JOINTURE, SELECTION, PROJECTION….)

- La représentation graphique de ces opérations permet de composer des arbres d'opérations relationnelles.

Exemple : soit la base de données composée des relations suivantes :

R1(MEDECIN , MALADIE , TARIF)
R2(NUMERO , MALADE , MALADIE)

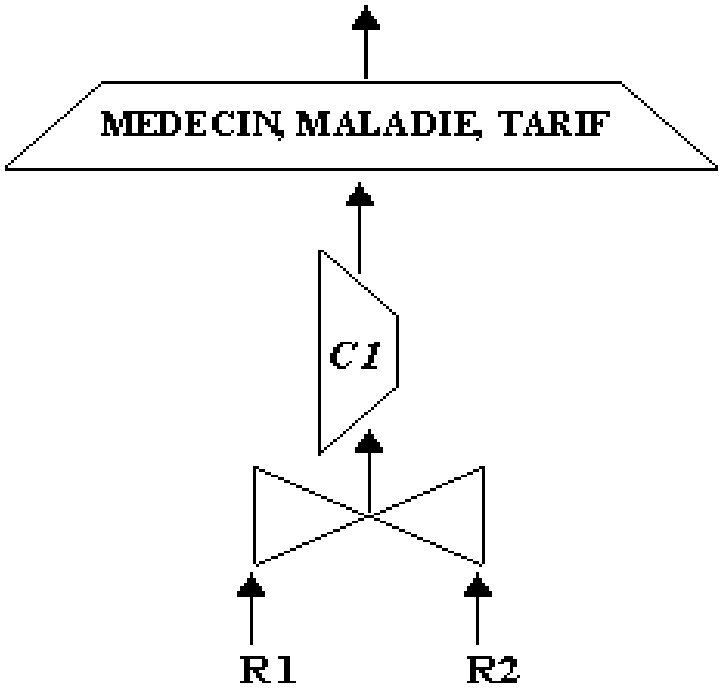
- La relation R1 représente les maladies qui peuvent être examinées par un médecin et le tarif de la consultation chez ce médecin pour chaque maladie.

- La relation R2 représente les maladies pour lesquelles un malade souhaite être examiné.

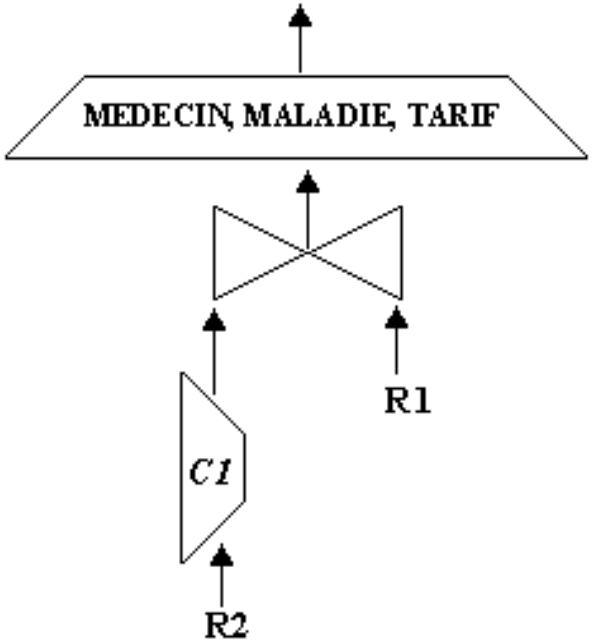
La réponse à la question suivante : "Quels sont les noms des médecins pouvant examiner le malade "RABAH" et les prix de leurs consultations ainsi que les maladies à examiner "

peut être exprimée à l'aide de l'un des deux arbres suivants :

Avec la condition C1 : Malade = "Rabah"



a)



b)

- Les expressions algébriques correspondant à chacun des deux arbres précédents sont respectivement :

a) Π médecin, maladie, tarif (σ malade = "RABAH" (**JOIN** (R1, R2)))

b) Π médecin, maladie, tarif (**JOIN** (σ malade = "RABAH" (R2), R1))

- b) est plus efficace que a) puisque on applique d'abord le SELECT sur R2 qui donnera une relation intermédiaire contenant uniquement les tuples relatifs au malade "RABAH" qu'on utilisera pour calculer le JOIN avec R1.
- Pour passer de a) à b) on utilise les propriétés des opérateurs (ici la P7 pour faire passer la sélection avant la jointure)

- **On remarque donc que la réponse à une même question peut s'obtenir de différentes façons.**

⇒ C'est à dire que différentes stratégies d'exécution d'une requête peuvent donner la réponse à la question.

- Parmi ces stratégies, certaines sont plus performantes que d'autres en terme de temps d'exécution et d'espace mémoire requis.

- Les SGBD relationnels intègrent tous **un module d'optimisation des requêtes** permettant de **choisir parmi plusieurs stratégies** possibles d'exécution d'une requête, celle qui est la plus performante.

- Cette optimisation est basée principalement sur l'application des propriétés des opérateurs de l'algèbre relationnelles.

SQL : l'essentiel

- Le langage SQL (Structured Query Language) est issu des travaux menés par IBM autour du langage prototype SEQUEL (Semi Query Language) pour le SGBD Système R.
- Aujourd'hui SQL est devenu un standard (normalisé par l'ANSI depuis 1986) disponible sur presque tous les SGBD relationnels (DB2, ORACLE, INFORMIX,...).
- Certains SGBD tel que INGRES offrent même deux types de langages, l'un algébrique (SQL) et l'autre prédicatif (QUEL).

❖ SQL supporte aussi bien les fonctions de description de données que celles de manipulation de données. La liste suivante permet de donner une idée sur les opérations qui peuvent être réalisées avec SQL :

- Créer le schéma de la base(Créer , supprimer une table ou un index, etc.)
- Modifier le schéma de la base (ajouter une nouvelle table, modifier le format d'une colonne, etc.)
- Interroger la base de données
- Définir des vues sur la base de données
- Spécifier les droits d'accès d'un utilisateur vis à vis des objets de la base (relations et vues)
- Mettre en place un mécanisme de contrôle d'intégrité des données
- Invoquer les commandes à partir d'un langage de programmation (Embedded SQL)

Structure d'une Requête en SQL

- Une requête d'interrogation en SQL est composée de trois clauses : **SELECT** , **FROM** et **WHERE** ayant chacune le rôle suivant :

- SELECT :
  - Cette clause **correspond à l'opération de projection de l'algèbre** relationnelle.
  - Elle est utilisée pour lister les attributs devant figurer dans le résultat d'une requête.
- FROM :
  - Cette clause **correspond à l'opération de produit cartésien** de l'algèbre relationnelle.
  - Elle est utilisée pour spécifier la liste des relations devant être utilisées pour évaluer l'expression.

- **WHERE :**
  - Cette clause correspond à la **condition ou qualification** utilisée avec l'opération de sélection de l'algèbre relationnelle.
  - Elle permet de spécifier une condition faisant intervenir des attributs appartenant aux relations figurant dans la clause FROM.

☝ Le terme **SELECT** utilisé dans une requête SQL **n'a rien à voir** avec celui utilisé dans l'opération de sélection de l'algèbre relationnelle notée elle aussi par **Select ou σ**.

☝ Dans la suite du texte nous utiliserons une notation en majuscules SELECT pour désigner la sélection dans une requête SQL et une notation en minuscules pour faire référence à l'opération algébrique de sélection Select.

➢ **Une requête SQL aura donc la forme suivante :**

SELECT A1 , A2 , ..., An
FROM R1 , R2 , ..., Rm
WHERE *C*

- Où chaque **Ai** représente un attribut, chaque **Ri** représente une relation et ***C*** représente une condition ou une qualification.

- Une telle requête est équivalente à l'expression de l'algèbre relationnelle suivante :

  – **Π A1 , A2 , ..., An (σC (R1 x R2 x .....x Rm))**

➢ Ainsi, on peut dire que l'évaluation de la requête va consister à

- calculer le produit cartésien entre les relations Ri figurant dans la clause FROM,
- puis à appliquer au résultat obtenu l'opération de sélection σC de l'algèbre relationnelle en utilisant la condition C figurant dans la clause WHERE,
- et enfin à appliquer au résultat obtenu l'opération de projection Π A1 , A2 , ..., An de l'algèbre relationnelle en utilisant la liste des attributs Ai figurant dans la clause SELECT de la requête.

☝ Néanmoins, il faut souligner que ce schéma d'évaluation est trop simpliste car dans la pratique, le SGBD peut être amené à convertir une requête en une autre requête équivalente mais plus performante en terme de temps d'exécution et ce en utilisant les propriétés des opérateurs de l'algèbre relationnelle.

❖ nous allons nous servir du schéma relationnel suivant :

EMPLOYES (Num_Emp, Nom_Emp, Fonction, Salaire, Prime, Num_Resp , Num_Dept)

DEPARTEMENTS (Num_Dept, Nom_Dept , Ville)

Expression de la projection avec SQL

- Une projection consiste à extraire des colonnes (attributs) spécifiques d'une relation puis à éliminer les tuples en double pouvant apparaître dans la relation résultat. Cette opération s'exprime a l'aide de SQL par la clause :

  SELECT ***liste d'attributs***
  FROM ***nom de relation***

- En pratique, SQL n'élimine pas les tuples en double car ces tuples ne gênent pas l'utilisateur et leur élimination implicite par SQL peut entraîner une perte de temps.

- Cependant SQL offre à l'utilisateur la possibilité de demander une élimination explicite de ces doubles grâce a un mot clé du langage qui est : **DISTINCT** (ou parfois **UNIQUE**).

Expression de la projection avec SQL

- A ce niveau, il est important de remarquer que **l'autorisation** de la présence de **tuples dupliqués** **est un inconvénient majeur de SQL** est qu'il n'est pas un langage ensembliste puisque ne fournissant pas automatiquement comme résultat un ensemble.
- Or nous avons vu que les opérations de l'algèbre relationnelle s'appliquaient à des relations et le résultat était aussi une relation.
- Ceci n'est en fait vrai que parce que les arguments de ces opérations sont supposées être des relations qui sont des ensembles au sens mathématique du terme c'est à dire ne contenant pas de tuples dupliqués.
- Dans le cas où une relation pouvait contenir un tuple en double, beaucoup de propriétés des opérateurs de l'algèbre relationnelle seraient remises en cause.

Q1: Donner les noms de tous les employés et le salaire de chacun d'eux ?

```
SELECT    Nom_Emp , Salaire
FROM      EMPLOYES
```

Q2: Liste de tous les Départements dans lesquels travaille au moins un employé?

```
SELECT DISTINCT   Num_Dept
FROM              EMPLOYES
```

On a rajouté le mot clé DISTINCT car le même numéro de Département peut apparaître plusieurs fois dans le résultat du fait que plusieurs Employés différents peuvent travailler dans un même département.

Q3: *Lister tous les tuples de la relation* **DEPARTEMENTS** *?*

```
SELECT    Num_Dept , Nom_Dept , Ville
FROM      DEPARTEMENTS
```

On voit que la liste des attributs après la clause **SELECT** inclut tous les attributs de la relation DEPARTEMENTS. Pour simplifier l'écriture d'une telle requête, SQL permet de les formuler comme suit :

```
SELECT    *
FROM      DEPARTEMENTS
```

ou le **SELECT *** signifie : " *Sélectionner tous les attributs de la relation* DEPARTEMENTS "

Expression de la sélection avec SQL

L'opération algébrique de sélection s'exprime dans SQL par un bloc du type :

SELECT *liste d'attributs*
FROM *nom de relation*
WHERE *condition*

Q4: ***Quels sont les employés travaillant dans le département numéro* 10**

```
SELECT*
FROM   EMPLOYES
WHERE Num_Dept = 10
```

Avec cette requête, on aura comme résultat une relation ayant le même schéma que la relation EMPLOYES (i.e. mêmes attributs) et contenant tous les tuples de la relation EMPLOYES pour lesquels l'attribut Num_Dept a pour valeur 10.

Q5: ***Quels sont les numéros et les noms des employés travaillant dans le département numéro* 10 ?**

```
SELECT Num_Emp, Nom_Emp
FROM   EMPLOYES
WHERE Num_Dept = 10
```

Le résultat de cette requête est identique a celui de la question Q4 sauf qu'ici on aura uniquement une relation formée de deux colonnes (attributs) qui sont **Num_Emp** et **Nom_Emp** et non pas une relation formée de tous les attributs comme c'était le cas pour **Q4** à cause de la clause **SELECT** *.

- On peut éventuellement ordonner le résultat d'une requête grâce à la clause **ORDER BY.** Pour cela, il suffit de spécifier dans cette clause le ou les attributs selon lesquels on désire ordonner le résultat ainsi que le critère ascendant ou descendant.

- La condition de la clause **WHERE** peut utiliser les opérateurs tels que: = , > , ≥ , < , ≤ , != , **AND** , **OR** , **NOT** , **BETWEEN** , **LIKE** , **IN** , **ANY**, **ALL**, **EXIST**, ... }

BETWEEN	: Test d'appartenance à un intervalle
IN	: Test d'appartenance d'une valeur à un ensemble
LIKE	: Test de ressemblance de chaînes de caractères
ANY	: Comparaison d'une valeur à une valeur quelconque d'un ensemble
ALL	: Comparaison d'une valeur à toutes les valeurs d'un ensemble
EXIST	: Test d'existence d'un tuple dans une relation (Quantificateur existentiel ∃)

Q6: ***Quels sont les numéros et les noms des employés du département numéro 10 et qui ont un salaire supérieur à 1000 ?***

```
SELECT Num_Emp, Nom_Emp
FROM   EMPLOYES
WHERE Num_Dept = 10  AND Salaire > 1000
```

Q7: ***Quels sont les fonctions exercées par les employés des départements 10 et 20 en éliminant les tuples en double du résultat?***

(a) : avec **IN**

```
SELECT  DISTINCT  Fonction
FROM              EMPLOYES
WHERE             Num_Dept  IN (10 , 20 )
```

(b): avec **OR**

```
SELECT DISTINCT  Fonction
FROM             EMPLOYES
WHERE            Num_Dept  = 10 OR Num_Dept  = 20
```


Q8: ***Quels sont les fonctions dont le salaire est compris entre 2000 et 3000?***

(a) : avec **BETWEEN**

```
SELECT  DISTINCT  Fonction
FROM              EMPLOYES
WHERE             Salaire
BETWEEN           2000 AND 3000
```

(b): avec **AND**

```
SELECT DISTINCT  Fonction
FROM             EMPLOYES
WHERE            Salaire >= 2000
AND              Salaire < = 3000
```

Q9: *Quels sont les employés du département numéro 10 dont le nom commence par la lettre* 'B' *?*

SELECT Nom_Emp

FROM EMPLOYES

WHERE Num_Dept = 10 **AND** Nom_Emp **LIKE** 'B%'

Le symbole % représente n'importe quelle chaîne de caractères. Il existe d'autres possibilités de comparaisons avec l'opérateur **LIKE**.

Par exemple, pour rechercher les employés dont le nom se termine par la lettre R , il suffit d'utiliser '%R' comme argument de **LIKE.**

Q10: ***Quels sont les employés du département numéro 10 dont la fonction n'est ni 'Ingénieur' ni 'Analyste' ?***

SELECT Nom_Emp

FROM EMPLOYES

WHERE Num_Dept = 10

AND NOT (Fonction = 'Ingénieur' **OR** Fonction = 'Analyste' **)**

Il est important à ce niveau de préciser que la réponse à une même question peut être obtenue avec une ou plusieurs requêtes différentes utilisant chacune un ou des opérateur(s) différent(s).

- Il est aussi possible d'exprimer en SQL les opérations algébriques UNION, PRODUIT CARTESIEN, JOINTURE , INTERSECTION , DIFFERENCE et DIVISION.

- Certaines opérations algébriques ne sont pas directement supportées par SQL car ce dernier ne proposant pas un opérateur spécifique pour les exprimer directement.

- Cependant, il existe toujours une possibilité pour les traduire en termes des opérateurs offerts par le langage.

- C'est le cas par exemple des opérations algébriques de JOINTURE , DIFFERENCE , INTERSECTION et DIVISION.

- Nous verrons comment les traduire en fonction des possibilités du langage dans le cas ou ce dernier ne les supporte pas directement.

Expression de l'union avec SQL

- L'opération d'union algébrique s'exprime dans SQL par un bloc du type :

SELECT *liste d'attributs*
FROM *nom de relation*
WHERE *condition*
UNION
SELECT *liste d'attributs*
FROM *nom de relation*
WHERE *condition*

Q11: **Quels sont les fonctions exercées par les employés des départements 10 et 20 en éliminant les tuples en double du résultat? (même que Q7)**

(a) : Pas de tuples en double

```
SELECT    Fonction
FROM      EMPLOYES
WHERE     Num_Dept = 10
UNION
SELECT    Fonction
FROM      EMPLOYES
WHERE     Num_Dept = 20
```

(b) : Avec tuples en double

```
SELECT    Fonction
FROM      EMPLOYES
WHERE     Num_Dept = 10
UNION ALL
SELECT    Fonction
FROM      EMPLOYES
WHERE     Num_Dept = 20
```

- Par défaut, l'opération d'**UNION** (requête (a)) élimine les tuples en double du résultat.
- Si on veut les garder, on doit utiliser **UNION ALL** (requête (b)) à la place de UNION tout court comme ci-dessus.
- Cette question étant la même que Q7, nous avons vu que dans la requête correspondant à Q7, on avait explicitement demandé l'élimination des tuples en double grâce au mot clé **DISTINCT**.

Expression du produit cartésien avec SQL

❖ Le produit cartésien entre deux relations est un cas particulier de jointure ou la **condition de jointure est absente** (i.e. clause WHERE).

• Par exemple le produit cartésien des relations EMPLOYES et DEPARTEMENTS s'obtient à l'aide de la requête suivante :

```
SELECT    *
FROM      EMPLOYES , DEPARTEMENTS
```

☝ Dans la définition de l'opération de produit cartésien, nous avons noté que cette opération ne peut être **évaluée que si les deux relations n'avait pas d'attributs en commun.**

- Dans le cas où les deux relations avaient des attributs en commun (c'est à dire ayant les même noms), **il fallait d'abord les renommer** pour être à même de calculer le produit cartésien de ces relations.
- Or dans notre exemple, les relations EMPLOYES et DEPARTEMENTS ont un attribut en commun à savoir Num_Dept. et pourtant nous ne l'avons pas renommé dans une de ces relations avant de calculer le produit cartésien.
- Ceci est dû au fait qu'avec SQL le renommage n'est pas nécessaire puisque ce dernier préfixe automatiquement un attribut avec le nom de sa relation selon la notation pointée Nom_de_Relation.Nom_Attribut.
- Ainsi le nom effectif de l'attribut Num_Dept de la relation EMPLOYES est EMPLOYES.Num_Dept et celui de la relation DEPARTEMENTS est DEPARTEMENTS. Num_Dept ce qui évite tout problème d'ambiguïté.

- Il est aussi tout à fait possible de préfixer dans une requête un attribut apparaissant dans une clause SELECT ou une clause WHERE avec le nom de sa relation comme dans la requête suivante (même que Q6) :

```
SELECT    EMPLOYES.Num_Emp, EMPLOYES.Nom_Emp
FROM      EMPLOYES
WHERE     Num_Dept = 10  AND Salaire > 1000
```

- Bien que cette écriture soit juste, aucun problème d'ambiguïté ne se pose dans cet exemple puisque on est sûr que les attributs Num_Emp et Nom_Emp appartiennent à la relation EMPLOYES. **Le préfixage n'est donc pas nécessaire.**
- Néanmoins, dans certaines requêtes, surtout celles mettant en jeu les jointures, on sera très souvent amené à utiliser cette notation afin de lever toute ambiguïté dans la désignation des attributs.

Expression de l'intersection avec SQL

- On sait que cette opération a comme arguments deux relations et donne comme résultat une relation contenant l'ensemble des tuples qui appartiennent en même temps à la première relation et à la deuxième. Un exemple de question illustrant cette opération est :

Q12: *Quels sont les numéros des départements dans lesquels travaillent des VENDEURS et des 'INGENIEURS'.*

```
SELECT      Num_Dept
FROM        EMPOYES
WHERE       Fonction  ='VENDEUR'
INTERSECT
SELECT      Num_Dept
FROM        EMPOYES
WHERE       Fonction  ='INGENIEUR'
```


Expression de la différence avec SQL

- On sait que cette opération a comme arguments deux relations et donne comme résultat une relation contenant l'ensemble des tuples appartenant à la première relation mais pas à la deuxième.

Q13: ***Quels sont les numéros des départements dans lesquels travaillent des 'VENDEURS' mais pas des 'INGENIEURS' ?***

```
SELECT      Num_Dept
FROM        EMPOYES
WHERE       Fonction  ='VENDEUR'
MINUS
SELECT      Num_Dept
FROM        EMPOYES
WHERE       Fonction  ='INGENIEUR'
```

- Si on se réfère à la définition de l'opération algébrique **R-S**, on remarque que la première relation R correspond au le résultat de la sous-requête :

```
SELECT    Num_Dept
FROM      DEPARTEMENT
WHERE     Fonction  ='VENDEUR'
```

qui contiendra les numéros de tous les départements dans lesquels travaille au moins un employé ayant pour fonction 'VENDEUR'

- alors que la deuxième relation S correspond au résultat de la sous-requête :

```
SELECT    Num_Dept
FROM      DEPARTEMENT
WHERE     Fonction  ='INGENIEUR'
```

qui contiendra les numéros de tous les départements dans lesquels travaille au moins un employé ayant pour fonction 'INGENIEUR'.

- D'après la définition de l'opération **MINUS**, le résultat sera l'ensemble des tuples appartenant à R mais pas à S.
- Donc si un même département possède au moins un employé ayant pour fonction 'VENDEUR' (figure dans R) et possède aussi au moins un employé ayant pour fonction 'INGENIEUR' (figure aussi dans S)

⇒alors ce département ne figurera pas dans le résultat de R-S, ce qui répond bien à la question.

Expression des jointures avec SQL

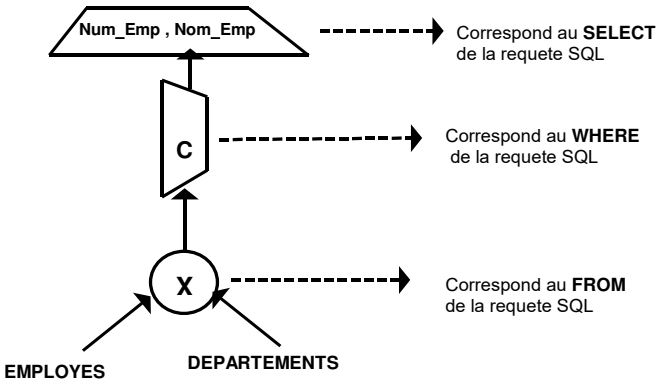
❖ Jointure avec qualification

- SQL n'offre pas une opération spécifique pour exprimer la jointure avec qualification ou avec condition.
- Comme cette opération algébrique est une opération complémentaire, elle peut s'exprimer en fonction des opérations de algébriques de base à savoir : la sélection et le produit cartésien.
- On sait que : **JOIN C (R,S) = Select C (R x S)** ou **C** représente la condition ou qualification de jointure faisant intervenir des opérateurs de comparaison.
- Il faudra donc **exprimer la jointure comme une sélection selon une condition** sur le produit cartésien des relations R et S .

Q14: Quels sont les numéros et les noms des employés qui travaillent à 'BATNA' ?

```
SELECT Num_Emp , Nom_Emp
FROM   EMPLOYES , DEPARTEMENTS
WHERE EMPLOYES•Num_Dept=DEPARTEMENTS•Num_Dept
AND    Ville = 'BATNA'
```

- :Une représentation graphique de l'expression algébrique permettant de répondre à cette question serait :



où la condition C est : EMPLOYES•Num_Dept=DEPARTEMENTS•Num_Dept
∧ Ville = 'BATNA'

Q15: Quels sont les noms ,les fonctions et les salaires des employés du département 'RECHERCHES' dont le salaire est supérieur à 1000 ?

```
SELECT  Nom_Emp , Fonction , Salaire
FROM    EMPLOYES , DEPARTEMENTS
WHERE   EMPLOYES•Num_Dept = DEPARTEMENTS•Num_Dept
        AND Nom_Dept = 'RECHERCHE'
        AND Salaire > 1000
```

- On remarque que le préfixage d'un attribut avec le nom de sa relation n'est utilisé que pour Num_Dept qui est commun aux deux relations participant dans la jointure.
- La condition spécifiée dans la clause WHERE correspond à la condition de jointure C.

➢ **Il est possible d'imbriquer des blocs** : SELECT ... FROM ...WHERE ..

à plusieurs niveaux.

Q16: **Quels sont les noms et les fonctions des employés travaillant à 'BATNA' et ayant la même fonction que l'employé 'AHMED' ?**

- Cette jointure peut s'exprimer sous forme **de blocs SELECT imbriqués.**

```
SELECT  Nom_Emp , Fonction
FROM    EMPLOYES , DEPARTEMENTS
WHERE   Ville = 'BATNA'
        AND EMPLOYES•Num_Dept = DEPARTEMENTS•Num_Dept
        AND Fonction IN
                ( SELECT    Fonction
                  FROM      EMPLOYES
                  WHERE     Nom_Emp = 'AHMED' )
```


Jointure Naturelle

- SQL n'offre pas une opération spécifique pour exprimer la jointure naturelle.
- Il faudra là aussi traduire en utilisant l'expression de la jointure naturelle en fonction des opérations algébriques de sélection , projection et produit cartésien et qui est :

JOIN (R,S) = $\Pi_V (\sigma_C (R \times S))$

ou v est égal l'union des attributs de R et des attributs de S n'appartenant pas à R ,

et la **condition C** : R.A1=S.A1 /\ R.A2=S.A2 /\..... /\ R.Ak=S.Ak avec (A1,A2,.....AK) les attributs communs à R et à S.

Jointure Naturelle

❖ L'expression de la jointure naturelle entre les relations EMPLOYES et DEPARTEMENTS qui ont un seul attribut commun peut s'écrire :

$$\Pi_{V} (\sigma_{C} (\text{EMPLOYES X DEPARTEMENTS}))$$

où :

V = (Num_Emp, Nom_Emp, Salaire, Fonction, Prime, Num_Dept, Nom_Dept, Ville)

et la condition de sélection **C** est :

EMPLOYES•Num_Dept = DEPARTEMENTS•Num_Dept

- d'où la requête SQL équivalente :


```
SELECT  Num_Emp,Nom_Emp,Salaire, Fonction, Prime, Num_Dept,Nom_Dept,Ville
FROM    EMPLOYES , DEPARTEMENTS
WHERE   EMPLOYES•Num_Dept = DEPARTEMENTS•Num_Dept
```

- Il n'est pas obligatoire de spécifier tous les attributs apparaissant dans la clause SELECT de la requête ci-dessus car **SQL ignore qu'il s'agit d'une jointure naturelle.**

- On peut même ne spécifier dans la clause SELECT qu'un sous ensemble d'attributs de la liste V seulement.

- Dans le cas où on utilise un SELECT * , on aura comme résultat une relation ayant pour schéma l'union des attributs de X1 et X2 (même schéma que la relation résultant du produit cartésien X1 x X2) c'est à dire tous les attributs de X1 plus tous les attributs de X2 indépendamment du fait que X1 et X2 ont les mêmes attributs.

- Rappelons que la notation pointée utilisée par SQL pour désigner un attribut X1.nom_Attribut ou X2.nom_Attribut fait que les attributs communs à deux relations de la base de données possèdent au niveau interne de SQL des noms distincts **à cause du préfixage et ce même si au niveau du schéma ils ont les mêmes noms.**

Equi-Jointure

- Cette opération n'existe pas aussi dans SQL.

- Cependant, il est possible de l'exprimer de la même manière que la jointure avec qualification puisque la seule différence est que dans l'équi-jointure l'opérateur de comparaison θ utilisé dans la condition de jointure est l'opérateur d'égalité (**=**).

Jointure d'une relation avec elle même (Auto-Jointure)

- Pour répondre à certaines questions, on peut parfois être amené à faire la **jointure d'une table avec elle-même**.
- Le problème qui se pose alors est comment distinguer les attributs.
- La solution qui est largement utilisée est de désigner dans la clause FROM de la requête SQL la relation (ou table) par deux noms différents appelés **synonymes**.
- Avec SQL, l'association d'un nom synonyme à une table consiste à **faire suivre le nom de la table par le nom synonyme** qui n'est autre qu'un identificateur au sens informatique du terme.

Q17: Quels sont les noms et les fonctions des employés ayant un salaire supérieur à celui de l'employé dont le nom est 'ALI'

```
SELECT   X1.Nom_Emp , X1.Fonction
FROM     EMPLOYES X1 , EMPLOYES X2
WHERE    X1.Salaire > X2.Salaire
AND      X2.Nomp_Emp = 'ALI'
```

- X1 et X2 sont les noms synonymes désignant la table EMPLOYES.
- Cette requête peut être vue comme une requête ayant pour arguments dans la clause FROM deux tables X1 et X2 ayant le mêmes attributs et la même extension que EMPLOYES.
- L'évaluation de la requête va consister à calculer **le produit cartésien entre les deux tables X1 et X2**,
  - puis de sélectionner les tuples du résultat pour lesquels la colonne X1.Salaire > X2.Salaire et la colonne X2.Nomp_Emp = 'ALI' .
  - Enfin, on fait une projection sur les colonnes X1.Nom_Emp et X1.Fonction qui sera alors la relation répondant à cette question.

➢ Ainsi, le schéma de la relation résultat du produit cartésien X1 x X2 sera

Attributs de la Table X1				Attributs de la Table X2		
X1•Num_Emp	X1•Nom_Emp	X1•Salaire	.....	X2•Num_Emp	X2•Nom_Emp	X2•Salaire

- Après sélection des lignes ou tuples satisfaisant la condition :
  X1•Salaire > **X2**•Salaire **AND X2**•Nomp_Emp = 'ALI' ,
- la projection permet de retenir les colonnes **X1**•Nom_Emp et **X1**•Fonction

Q18: Quels sont les noms des employés ayant le même responsable ?

```
SELECT   EMP1•Nom_Emp , EMP2•Nom_Emp
FROM     EMPLOYES  EMP1 , EMPLOYES EMP2
WHERE    EMP1•Num_Resp =  EMP2•Num_Resp
```

- Dans cette requête EMP1 et EMP2 sont les noms synonymes de la table EMPLOYES.
- Les étapes d'évaluation de la requête sont les mêmes que celles de **Q17**.

- Cependant, il faut remarquer ici que dans le résultat on peut avoir **des tuples en double** et des **tuples symétriques** traduisant les situations qui suivent :
  - Dans notre exemple, les employés Tahar et Ahmed ont le même responsable (Num_Resp = 4 qui correspond à l'employé Omar).
  - Dans le résultat de notre requête, on aura une relation ayant deux attributs et contenant parmi ses tuples les suivants :

EMP1•Nom_Emp	EMP2•Nom_Emp
Tahar	Tahar
Tahar	Ahmed
Ahmed	Ahmed
Ahmed	Tahar

- Les tuples **2 et 4** traduisent la même information : *Tahar et Ahmed ont le même responsable.* Il faudrait donc ne garder qu'un seul de ces deux tuples.
- Les tuples **1 et 3** traduisent le fait que tout employé a le même responsable que lui-même. Ils n'apportent aucune information et doivent être éliminés du résultat.

- Pour traiter ces cas il suffit de **rajouter dans la clause WHERE** une condition supplémentaire :

EMP1•Nom_Emp < EMP2•Nom_Emp (ou >)

```
SELECT   EMP1•Nom_Emp , EMP2•Nom_Emp
FROM     EMPLOYES EMP1 , EMPLOYES EMP2
WHERE    EMP1•Num_Resp = EMP2•Num_Resp
         AND EMP1•Nom_Emp < EMP2•Nom_Emp
```

- L'application de cette condition a lieu au moment de la sélection des lignes à partir de la relation résultant du produit cartésien entre EMP1 et EMP2.
- Voyons quelles sont parmi les lignes qui contenaient dans les colonnes EMP1.Nomp_Emp et EMP2.Nom_Emp les valeurs **Tahar et Ahmed** celles qui seront retenues par l'opération de sélection :

- **Tahar n'est pas < à Tahar** : donc la ligne qui contient dans la colonne EMP1.Nom_Emp la valeur Tahar et dans la colonne EMP2.Nom_Emp la valeur Tahar aussi ne figurera pas dans la relation résultat de la sélection
- **Tahar n'est pas < à Ahmed** : donc la ligne qui contient dans la colonne EMP1.Nom_Emp la valeur Tahar et dans la colonne EMP2.Nom_Emp la valeur Ahmed ne figurera pas dans la relation résultat de la sélection.
- **Ahmed n'est pas < à Ahmed** : donc la ligne qui contient dans la colonne EMP1.Nom_Emp la valeur Ahmed et dans la colonne EMP2.Nom_Emp la valeur Ahmed aussi ne figurera pas dans la relation résultat de la sélection.
- **Ahmed est < à Tahar** : donc la ligne qui contient dans la colonne EMP1.Nom_Emp la valeur Ahmed et dans la colonne EMP2.Nom_Emp la valeur Tahar figurera dans la relation résultat de la sélection

➢ Donc après projection on aura uniquement le tuple (Ahmed,Tahar) dans le résultat final et correspondant au fait que Tahar et Ahmed ont le même responsable. **Il en sera de même pour les autres employés ayant le même responsable.**

Utilisation des sous-requêtes

- Le terme sous-requête désigne toute requête utilisée à l'intérieur d'une requête dite principale ou externe.
- Elle retourne comme toute requête un résultat qui est une relation qui peut comporter une ou plusieurs colonnes et aussi une ou plusieurs lignes
- La sous requête sert en général pour comparer un attribut ou un ensemble d'attributs de la requête principale au résultat retourné par la sous requête.
- Il faut remarquer au passage que les sous requêtes **peuvent aussi être utilisées** dans les opérations **d'insertion**, de mise à jour et de **suppression** de tuples dans une table.

❖ Sous-requête retournant une seule valeur

Q19: **Quels sont les noms des employés ayant la même fonction que l'employé 'OMAR'**

```
SELECT    Nom_Emp
FROM      EMPLOYES
WHERE     Fonction =
              ( SELECT    Fonction
                FROM      EMPLOYES
                WHERE     Nomp_Emp  ='OMAR' )
```

- La sous requête doit être placée entre parenthèses.
- Elle est évaluée en premier et va retourner une seule valeur 'DIRECTEUR'.
- Cette valeur sera utilisée pour constituer la condition de sélection de la clause WHERE de la requête principale qui sera donc :

Fonction = 'DIRECTEUR'.

❖ Sous-requête retournant un ensemble de valeurs

- Lorsqu'une sous requête retourne un ensemble de valeurs ,

- il faudra **faire précéder** l'opérateur de comparaison (= , != , > , < , <= , >=) de la clause WHERE de la requête principale avec un des **mots clefs ANY ou ALL**.

Q20: **Quels sont les noms des employés ayant un salaire supérieur à celui d'un employé quelconque du département 30 ?**

```
SELECT    Nom_Emp
FROM      EMPLOYES
WHERE     Salaire > ANY
          ( SELECT    Salaire
            FROM      EMPLOYES
            WHERE     Nump_Dept  = 30 )
```

- La sous requête va retourner un ensemble de valeurs représentant les salaires du département 30.

- Pour chaque employé, la requête principale va comparer son salaire avec l'ensemble des valeurs retournées par la sous requête.

- Si le salaire est supérieur à une valeur quelconque de cet ensemble, cet employé sera inclus dans le résultat.

Q21: **Quels sont les noms des employés ayant un salaire supérieur à celui de tous les employés du département 30 ?**

```
SELECT      Nom_Emp
FROM        EMPLOYES
WHERE       Salaire > ALL
                ( SELECT      Salaire
                  FROM        EMPLOYES
                  WHERE       Nump_Dept  = 30 )
```

On peut dans le cas ou la condition est = ANY , remplacer ce test par l'opérateur **IN**.

- De même que != ALL peut être remplacer par **NOT IN**.

Q22: **Quels sont les noms et les fonctions des employés du département 10 ayant la même fonction qu'un employé quelconque du département 30 ?**

```
SELECT      Nom_Emp , Fonction
FROM        EMPLOYES
WHERE       Num_Dept = 10
AND         Fonction IN
                ( SELECT      Fonction
                  FROM        EMPLOYES
                  WHERE       Nump_Dept  = 30 )
```

❖ Sous-requête retournant plusieurs colonnes

- Une sous requête peut retourner plus d'une colonne si la liste des attributs de sa clause SELECT comprend plus d'un attribut.
- Le nombre d'attributs de la requête principale à comparer avec l'ensemble des lignes (ou valeurs) retournées par la sous requête **doit être égale** au nombre de **colonnes** retournées par la **sous requête**.

Q23: **Quels sont les employés ayant le même salaire et la même fonction que l'employé 'Omar' ?**

```
SELECT    *
FROM      EMPLOYES
WHERE     (Salaire , Fonction) =
                    ( SELECT    Salaire , Fonction
                      FROM      EMPLOYES
                      WHERE     Nom_Emp = 'Omar' )
```


- Il est aussi possible **d'utiliser plusieurs sous requêtes imbriquées** et construire ainsi des requêtes aussi complexes qu'on le souhaite. C'est ce qui fait la puissance du langage SQL.
- Le nombre autorisé de sous requêtes imbriquées dépend de l'implémentation du langage. A titre d'exemple les premières versions du SGBD ORACLE (V3 et V4 datant de 1982-1983) autorisait **jusqu'à 16 sous requêtes imbriquées** en plus de la requête principale.

Q24: **Quels sont les noms et les fonctions des employés ayant la même fonction que l'employé 'Omar' ou un salaire supérieur ou égal à celui de 'Rachid'?**

```
SELECT   Nom_Emp , Fonction
FROM     EMPLOYES
WHERE    Fonction =
                    ( SELECT   Fonction
                      FROM     EMPLOYES
                      WHERE    Nom_Emp = 'Omar' )

                    OR         Salaire > =

                    ( SELECT   Salaire
                      FROM     EMPLOYES
                      WHERE    Nom_Emp = 'Rachid' )
```


❖ Utilisation des sous requêtes dans une Jointure

- Une sous requête peut être aussi utilisée dans l'expression d'une jointure.
- L'évaluation de la sous requête dépend de la façon dont on structure la requête qui la contient.
- Les exemples suivants permettent de donner une idée sur quelques possibilités.

Q25: **Quels sont les noms, Salaires et fonctions des employés travaillant à 'Alger' et ayant la même fonction que 'Rachid'?**

```
SELECT  Nom_Emp , Fonction , Salaire
FROM    EMPLOYES , DEPARTEMENTS
WHERE   Ville  = 'Alger'
AND     EMPLOYES•Num_Dept = DEPARTEMENTS•Num_Dept
AND     Fonction IN
            ( SELECT    Fonction
              FROM      EMPLOYES
              WHERE     Nom_Emp  = 'Rachid' )
```

- Dans cette jointure, la sous requête est exécutée en premier et fournit comme résultat un ensemble de valeurs correspondant aux fonction des employés ayant pour nom 'Rachid'.

- Si on est sûr que deux employés ne peuvent pas avoir le même nom **alors le IN peut être remplacé par =.**
- Le résultat est utilisé pour constituer la partie de la condition de jointure :

  Fonction IN <Ensemble des valeurs retournées par la sous requête>.

- D'après l'exemple , l'ensemble retourné par la sous requête étant égal à 'Vendeur', la condition de jointure serait équivalente à :

```
Ville  = 'Alger'
AND EMPLOYES•Num_Dept = DEPARTEMENTS•Num_Dept
AND Fonction IN ('Vendeur' )
```

➢ Nous allons voir un autre exemple où l'évaluation de la sous requête ne se fait pas en une seule fois indépendamment de la requête principale mais est répétée plusieurs fois en fonction de certaines données provenant de la requête principale.

Q26: **Quels sont les noms et Salaires des employés qui ont un salaire supérieur à la moyenne des salaires de leur départements?**

❑ L'obtention d'une réponse à cette question va nécessiter les étapes suivantes :

- Parcourir la table EMPLOYES de façon à connaître le numéro **d** du département de l'employé et son salaire **s**
- Calculer la moyenne **md** des salaires du département **d** (dans une sous requête)
- Tester si le salaire **s est > md** et si oui inclure l'employé dans le résultat

- Une requête permettant de répondre à cette question serait :

```
SELECT  Nom_Emp , Salaire
FROM    EMPLOYES , X
WHERE   Salaire >
                ( SELECT        AVG(Salaire)
                  FROM          EMPLOYES
                  WHERE         X•Num_Detp  = Num_Dept )
```

- On remarque l'utilisation d'un nom **synonyme X** pour la table EMPLOYES dans la clause FROM de la requête principale.

- **X** peut être vu comme une variable qui parcourt les tuples de la relation EMPLOYES apparaissant dans la requête principale.

- La clause WHERE X•Num_Detp = Num_Dept de la sous requête lui permet en quelque sorte de **se synchroniser avec la requête principale**.

- Ainsi pour chaque tuple repéré par la variable X , la sous requête va calculer la moyenne (fonction AVG : Average) des salaires du département ayant le même numéro que celui du **tuple repéré par X** et qui correspond à un employé bien sûr.

- Le fait que la sous requête référence la même table EMPLOYES dans sa clause FROM ne gêne en aucun cas la requête principale.

❖ Autres possibilités d'interrogation de SQL

➢ En logique on utilise généralement les deux quantificateurs □ ∃ (il existe) et □ ∀ (quelque soit) qui s'appliquent à des propositions selon les deux formes :

□ ∃ x P(x)

et ∀ x P(x) .

- Dans le premier cas la P(x) est vraie s'il existe au moins une valeur de x qui la rend vraie

- et dans le deuxième cas P(x) est vraie si et seulement si elle est vraie pour toutes les valeurs de x .

➢ **SQL** offre un prédicat qui s'appelle **EXISTS** équivalent du quantificateur existentiel ∃.

- Il permet de tester si l'ensemble de valeurs (tuples) retourné par une requête (ou sous requête) est vide ou non.
- Ce test est évalué à vrai (TRUE) si l'ensemble contient au moins une ligne et à faux (FALSE) si l'ensemble retourné est vide.

☝ **SQL n'offre pas** de prédicat équivalent du quantificateur ∀.

- Cependant, ce quantificateur **peut être traduit** en utilisant l'opérateur EXISTS.
- En effet, Il suffit de remarquer l'équivalence entre les deux formules suivantes :

$$\forall x\ P(x) \equiv \neg \exists x \neg P(x)$$

- L'utilisation du quantificateur ∀ est très rare dans l'expression des requêtes.
- Il **sert surtout** pour **exprimer** l'opération algébrique de **DIVISION** dont la définition formelle utilise justement ce quantificateur.

❑ Expression de la différence avec EXISTS

➢ La différence entre deux relations peut s'exprimer grâce a l'opérateur **MINUS** dont l'utilisation est semblable a celle de l'union entre deux relations.

➢ Il est aussi possible **d'utiliser le prédicat EXISTS** pour réaliser cette opération.

Q27: **Quels sont les numéros des départements dans lesquels travaillent des 'VENDEURS' mais pas des 'INGENIEURS' ?**

```
SELECT    Num_Dept
FROM      DEPARTEMENT
WHERE     Fonction  ='VENDEUR'
AND       NOT EXISTS
              ( SELECT    Num_Dept
                FROM      DEPARTEMENT
                WHERE     Fonction  ='INGENIEUR' )
```


❖ On sait que la division d'une relation R de schéma R(A,B,C) par une relation S de

schéma S(C) est une relation Q de schéma Q(A, B) et telle que :

Q = { (a, b) ∈ à R / ∀ (c) ∈ à S , ∃ (a,b,c) ∈ à R) }

- en utilisant l'équivalence des formules : **∀ x P(x) ≡ ⌉∃ x ⌉P(x)**

on peut traduire la division comme suit :

(a,b) ∈ Q si et seulement si **il n'existe pas** de **c ∈ à S**
tel qu'il n'existe pas de (a,b,c) ∈ R.

Ainsi, la division va se traduire en SQL par l'utilisation de deux opérateurs NOT EXITS successifs.

- Nous allons illustrer la **mise en ouvre de la division** à l'aide des deux relations suivantes :

  PROFESSEURS(Num_Prof, Code_Mod)

  MODULES(Code_Mod , Spécialité)

- Pour répondre à la question suivante :

Q28: **Quels sont les numéros des enseignants qui dispensent TOUS les modules de la spécialité 'INFORMATIQUE'?**


```
SELECT   DISTINCT (Num_Prof)
FROM     PROFESSEURS X
WHERE    NOT EXISTS
          ( SELECT    *
          FROM        MODULES
          WHERE       MODULES.Spécialité ='INFORMATIQUE'
          AND         NOT EXISTS
                       ( SELECT   *
                       FROM       PROFESSEURS
                       WHERE      MODULES•Code_Mod=PROFESSEURS•Code_Mod
                       AND        PROFESSEURS•Num_Prof = X•Num_Prof ) )
```

Les fonctions de groupes

- SQL offre aussi un certain nombre de fonctions dites fonctions de groupes qui peuvent être utilisées dans l'expression d'une requête. les plus importantes sont :
  - **AVG** : permet de calculer une MOYENNE
  - **SUM** : permet de calculer une SOMME
  - **COUNT** : permet de compter des tuples
  - **MAX** : permet de calculer un maximum
  - **MIN** : permet de calculer un minimum
  - **GROUP BY** : permet de créer des sous-ensembles de tuples
  - **HAVING** : permet de tester si une condition est vérifiée par un groupe de tuples
- Nous allons commencer par illustrer à l'aide d'exemples comment utiliser les fonctions AVG, MIN , MAX et COUNT puis nous examinerons de plus près la manipulation des groupes de tuples.

Q29: Quels sont les numéros et salaires des employés dont le salaire est supérieur à 10% de la moyenne des salaires de son département ?

```
SELECT    Nom_Emp , Salaire
FROM      EMPLOYES , X
WHER      Salaire >
              ( SELECT    AVG(Salaire) * 0.10
                FROM      EMPLOYES
                WHERE     X•Num_Detp  = Num_Dept )
```

Q30: **Quel est le nom , la fonction et le salaire de(s) l'employé(s) ayant le salaire le plus élevé**

```
SELECT    Nom_Emp , Fonction , Salaire
FROM      EMPLOYES
WHERE     Salaire =
              ( SELECT    MAX(Salaire)
                FROM      EMPLOYES)
```

- La sous requête est évalué en premier et permet de calculer le salaire maximum des employés.
- La requête principale va parcourir les lignes de la table EMPLOYES et comparer le salaire de chaque employé avec le résultat de la sous requête.
- S'il y a égalité, le nom , la fonction et le salaire de cet employé figurera dans le résultat final.

Q31: **Quel est le salaire maximum , le salaire minimum et la différence entre ces deux valeurs ?**

```
SELECT    MAX(Salaire), MIN(Salaire) , MAX(Salaire) - MIN(Salaire)
FROM      EMPLOYES
```

Q32: Quel est le nombre d'employés percevant une prime ?

```
SELECT    COUNT(Prime)
FROM      EMPLOYES
```

Q33: **Quel est le nombre de fonctions différentes exercées dans le département 30 ?**

```
SELECT   COUNT(DISTINCT  Fonction)
FROM     EMPLOYES
WHERE    Num_Dept  = 30
```

- Le mot **DISTINCT** clé permet de ne compter une même valeur qu'une seule fois.
- Le nombre retourné sera bien le nombre de valeurs différentes de l'attribut Fonction.

❖ Manipulation de groupes : la clause GROUP BY

- La clause **GROUP BY** permet de **partitionner les tuples d'une relation** de façon à former des groupes de tuples.
- Chaque groupe de tuples est caractérisé par le fait que les tuples qu'il contient possèdent les mêmes caractéristiques.
- La manipulation des groupes implique donc que les requêtes peuvent comporter des clauses WHERE , GROUP BY, des fonctions de groupes (AVG, SUM, MAX, MIN) et HAVING.
- La **clause WHERE** est utilisée pour sélectionner dans une table les lignes (tuples) qui satisfont une condition. **Elle ne s'applique donc pas aux groupes**.
- La **clause HAVING** est utilisée pour sélectionner dans une table les groupes de lignes (tuples) qui satisfont une condition. **Elle ne s'applique donc pas aux lignes** et **s'utilise avec une clause GROUP BY**.

☞ **Dans une requête** manipulant des fonctions de **groupe**, **il existe un ordre logique d'exécution de la requête et qui est :**

1- La clause **WHERE** est appliquée en premier pour qualifier les lignes

2- Les groupes de lignes sont formés (**GROUP BY**)

3- Les **fonctions de groupes** sont appliquées(AVG, MIN , MAX,..)

4- Enfin la clause **HAVING** est appliquée pour choisir les groupes répondant au critère spécifié dans cette clause.

Q34 : Quelle est la moyenne des salaires du département 10 ?

```
SELECT AVG (Salaire)
FROM   EMPLOYES
WHERE  Num_Dept  = 10
```

- Si on veut connaître la moyenne des salaires des départements 20 et 30, il suffit de refaire la même requête en remplaçant dans la clause WHERE

  la valeur 10 par 20 puis par 30

➢ **Au lieu d'utiliser 3 requêtes séparées**, on peut obtenir la moyenne des salaires par département avec une seule requête **en utilisant une clause GROUP BY**

```
SELECT      Num_Dept , AVG (Salaire)
FROM        EMPLOYES
GROUP BY    Num_Dept
```

Q35: **Quels est pour chaque département et chaque fonction le salaire annuel (12 mois) moyen par fonction et le nombre d'employés exerçant cette fonction?**

```
SELECT      Num_Dept, Fonction , Count (*) , AVG (Salaire) * 12
FROM        EMPLOYES
GROUP BY    Num_Dept , Fonction
```

Q36: **même question que Q35 mais donner le nom du département au lieu du numéro ce qui va nécessiter de faire une jointure ?**

```
SELECT      Nom_Dept, Fonction , Count (*) , AVG (Salaire) * 12
FROM        EMPLOYES , DEPARTEMENTS
WHERE       EMPLOYES•Num_Dept= DEPARTEMENTS•Num_Dept
GROUP BY    Num_Dept , Fonction
```


Q37: **Quels est pour chaque département le salaire annuel (12 mois) moyen de tous ses employés sauf ceux ayant une fonction de 'DIRECTEUR' ou 'PRESIDENT'?**

```
SELECT      Num_Dept, AVG (Salaire) * 12
FROM        EMPLOYES
WHERE       Fonction NOT IN ('DIRECTEUR' , 'PRESIDENT')
GROUP BY    Num_Dept
```

Q38: Quels sont les numéros des départements ayant plus de 10 employés ?

```
SELECT      Num_Dept
FROM        EMPLOYES
GROUP BY    Num_Dept
HAVING      COUNT(*) > 10
```

- Le COUNT(*) compte les tuples des sous-ensembles créés par GROUP BY.
- Les tuples de chaque sous-ensemble créé ont la même valeur de l'attribut qui est utilisé dans le GROUP BY (c.a.d. ici Num_Dept).
- Le résultat de cette requête sera une relation partitionnée en groupes de tuples ayant la même valeur de l'attribut Num_Dept.
- A chacun de ces groupes sera appliquée la qualification de la clause HAVING c'est a dire COUNT(*) > 10.
- Si le groupe satisfait la condition (contient plus de 10 tuples), il sera inclut dans le résultat par le biais de son attribut Num_Dept jouant le rôle de dénominateur commun pour les tuples du groupe.

Q39: Quelles sont les différentes fonctions exercées dans l'ensemble des départements et la moyenne des salaires par fonction ?

```
SELECT      Fonction , AVG (Salaire)
FROM        EMPLOYES
GROUP BY    Fonction
```

Q40: Quelles sont les différentes fonctions exercées dans l'ensemble des départements dont la moyenne des salaires par fonction est supérieure à 10.000 ?

```
SELECT      Fonction , AVG (Salaire)
FROM        EMPLOYES
GROUP BY    Fonction
HAVING      AVG (Salaire) > 10.000
```

Q41: **Quels sont les numéros des départements ayant au moins deux employés exerçant la fonction de 'Vendeur'?**

```
SELECT      Num_Dept
FROM        EMPLOYES
WHERE       Fonction = 'Vendeur'
GROUP BY    Num_Dept
HAVING      COUNT(*) > 2
```

Q42: **Quels sont les numéros des départements dans lesquels la moyenne des primes est supérieur à 10% de la moyenne des salaires?**

```
SELECT      Num_Dept, AVG (Salaire),AVG (Prime) , AVG (Salaire) * 0.10
FROM        EMPLOYES
GROUP BY    Num_Dept
HAVING      AVG (Prime) >  AVG (Salaire) * 0.10
```


Dans la liste des attributs figurant dans une clause SELECT si on a des noms d'attributs avec fonctions de groupe (AVG , MAX, MIN , ...), on ne peut pas avoir en même temps de nom d'attribut sans fonction de groupe.

Par exemple **l'écriture suivante serait une erreur** :

SELECT Nom_Emp , AVG (Salaire)

- parce que l'attribut Nom_Emp est un attribut ayant une valeur dans chaque ligne alors que AVG (Salaire) représente une valeur correspondant à un ensemble de lignes qui seront sélectionnées.

- Si on veut combiner dans la liste un attribut sans fonction de groupe avec un autre avec fonction de groupe, **on doit utiliser une sous requête.**

Tri du résultat d'une requête : la clause ORDER BY

- La clause ORDER BY permet de trier le résultat retourné par une requête.
- L'ordre peut être ascendant (ASC) ou descendant (DESC) et sera fonction du type de ou des attributs qui seront spécifiés comme arguments de cette clause.
- Les exemples suivants permettent de donner une idée des possibilités de cette clause.

Q43: Liste des employés du département 10 par ordre décroissant de leur salaire?

```
SELECT    Num_Dept , Nom_Emp, Salaire
FROM      EMPLOYES
WHERE     Num_Dept = 10
ORDER BY  Salaire DESC
```

Q44: Liste par ordre alphabétique des employés du département 10?

```
SELECT    Nom_Emp
FROM      EMPLOYES
WHERE     Num_Dept = 10
ORDER BY  Nom_Emp
```

- Il est aussi possible de trier le résultat retourné par une requête en fonction de plusieurs attributs.

Q45: Liste des employés triés selon un ordre alphabétique de leur fonction et a l'intérieur de chaque fonction les trier selon un salaire décroissant.

```
SELECT      Nom_Emp, Fonction, Salaire
FROM        EMPLOYES
ORDER BY    Fonction ASC, Salaire DESC
```

- Cette requête va entraîner dans un premier temps un tri du résultat par ordre alphabétique selon la fonction des employés.

- Puis un autre tri sera effectué sur les tuples correspondant aux employés **ayant la même fonction** de façon a les arranger selon **un ordre décroissant de leur salaire.**

❖ La description des données avec SQL

- SQL offre un certain nombre de commandes pour créer des tables et pour modifier leurs structures. Les principales commandes sont :

CREATE TABLE	: ajouter une table à la B.D.
ALTER TABLE	: modifier une colonne ou ajouter une nouvelle colonne à une table
DROP TABLE	: supprimer une table

❖ Création de tables : La commande CREATE TABLE

- Pour créer une table on spécifie le nom de la table, les noms des colonnes ainsi que le type des données de chaque colonne. Un nom de table ou de colonne (attribut) est un identificateur au sens informatique du terme et obéit en général aux règles syntaxiques de construction d'un identificateur (i.e. commencer par un caractère alphabétique, etc.).

- Les Types de données possibles pour un attribut sont variés (dépend du SGBD) :

Exemples :

CHAR	: caractères
NUMBER	: chiffre de 0 à 9,signe et point décimal
DATE	: date (MM-JJ-AA)

- On peut aussi spécifier la longueur maximal d'un champ (CHAR ou NUMBER).

Exemple:

CHAR(12)	: maximum 12 caractères
NUMBER(4)	: 4 chiffres au maximum
NUMBER(7,2)	: maximum 7 chiffres dont 2 à droite du point décimal

- La spécification de la longueur maximale d'un champ permet au SGBD de contrôler si une valeur ajoutée dans la colonne ne dépassera pas ce maximum.

- Pour spécifier qu'une colonne ne doit pas contenir de valeurs NULL (ou indéfinies, inconnues), il faut ajouter après le type **l'option NOT NULL.**

Exemple:

```
CREATE TABLE  ETUDIANTS ( NUMERO      NUMBER  NOT NULL,
                          NOM         CHAR(10),
                          MOYENNE     NUMBER(4,2) )
```

- L'attribut NUMERO est un nombre dont la taille sera prise par défaut et ne comportera de valeurs NULL.
- L'attribut NOM est une chaîne de 10 caractères au maximum.
- L'attribut MOYENNE est un nombre de 4 chiffres dont 2 après la virgule.

Extension d'une colonne: La commande ALTER TABLE

- La taille d'une colonne peut être augmentée grâce à la commande ALTER TABLE dont le format est :

```
ALTER TABLE     <nom de table>
MODIFY          (<nom de colonne>,<type(nouvelle taille)>)
```

Exemple:

```
ALTER TABLE     ETUDIANTS

MODIFY          (MOYENNE,NUMBER(6,2))
```

❖ Ajout d'une nouvelle colonne : La commande ALTER TABLE

- L'ajout d'une nouvelle colonne peut se faire grâce à la commande ALTER TABLE avec le Format suivant :

```
ALTER TABLE     <nom de table>
ADD             (<nom de colonne>,<type>)
```

Exemple:

```
ALTER TABLE     ETUDIANTS
ADD             (AGE,NUMBER(2))
```

- La nouvelle colonne AGE sera rajoutée à droite des autres colonnes de la table et sera initialisée par des valeurs NULL.
- On ne doit donc pas spécifier l'option NOT NULL à ce niveau.
- Cette colonne peut être remplie grâce à la commande **UPDATE**

❖ Suppression d'une table : DROP TABLE

- Pour supprimer une table, on utilise la commande **DROP TABLE** ayant le format suivant :

```
DROP TABLE      <nom de table>
```

Exemple:

```
DROP TABLE      ETUDIANTS
```

qui va provoquer la suppression de la table ETUDIANTS.

❖ Requêtes de mise a jour d'une table

- SQL offre les 3 commandes suivantes:

INSERT : ajouter une ligne dans une table

UPDATE : changer une valeur dans une ligne de la table

DELETE : supprimer une ligne dans une table

- Pour les exemples, On supposera pour la suite que la relation **ETUDIANTS**
- a pour schéma :

ETUDIANTS(NUMERO, NOM, MOYENNE, AGE)

Insertion de lignes dans une table : la commande INSERT

- La commande INSERT permet d'insérer des lignes (tuples) dans une table.
- Son format général est :

INSERT INTO <nom de relation>

VALUES (liste de valeurs)

Exemple:

- On veut insérer un tuple correspondant à l'étudiant ayant pour numéro 795 , pour nom JOJO , pour moyenne 12.54 et pour âge 21 ans.

INSERT INTO ETUDIANTS

VALUES (795,'JOJO',12.54,21)

- On peut ajouter une ligne incomplète en ne donnant que les valeurs de certains attributs. Pour cela, il faut préciser les noms de ces attributs (i.e. colonnes) dans INSERT.

Exemple:

- On veut insérer un tuple correspondant à l'étudiant ayant pour numéro 15 , pour nom TITI , pour âge 19 ans et dont on ne connaît pas le moyenne.

```
INSERT INTO  ETUDIANTS(AGE,NUMERO,NOM)
VALUES       (15, 'TITI', 19)
```

- Toutes les autres colonnes de la table seront initialisées avec la valeur NULL.

☞ On remarque que l'ordre des colonnes n'a pas d'importance alors que dans l'exemple précédent si car on n'a pas spécifié les noms des attributs et donc l'ordre implicite est celui donné lors de la création de la table.

Mise à jour de lignes dans une table : UPDATE

- La commande UPDATE permet de changer la valeur d'un attribut. Son Format général est :

```
UPDATE    <nom de table>
SET       <nom de colonne> = valeur
WHERE     <liste de conditions>
```

- La clause WHERE est optionnelle. Elle sert pour faire des changements sélectifs sur les lignes satisfaisant les conditions spécifiées.
- WHERE peut contenir les mêmes conditions de recherches que celles possibles avec une clause SELECT.

Exemple 1 : **Mise à jour d'une seule colonne d'une ligne**

- Ajouter 0.5 a la moyenne de L'étudiant ayant pour nom TOTO ,

```
UPDATE      ETUDIANTS
SET         MOYENNE = MOYENNE + 0.5
WHERE       NOM     = 'TOTO'
```

Exemple 2 : **Mise à jour de plusieurs colonnes dans une même ligne**

- des erreurs de saisies se sont glissées dans l'âge, la moyenne et le nom de l'étudiant numéro 795. Il faudra alors les corriger :

```
UPDATE      ETUDIANTS
SET         AGE=25 , NOM = 'OMAR' , MOYENNE =09.5
WHERE       NUMERO = 795
```

- Les colonnes doivent être séparées par une virgule dans la clause SET.

Suppression de lignes dans une table : DELETE

- La commande DELETE permet de supprimer des lignes d'une table. Elle permet donc de supprimer une seule ligne dans une table ou plusieurs en même temps. Son format général est :

```
DELETE  FROM    <nom de relation>
WHERE           <liste de conditions>
```

- La clause WHERE est optionnelle. Elle sert à faire des suppressions sélectives des lignes satisfaisant les conditions spécifiées.

- WHERE peut contenir les mêmes conditions de recherches que celles possibles avec SELECT ou UPDATE.

Exemple 1:

Supprimer l'étudiant ayant pour nom : TOTO

```
DELETE FROM     ETUDIANTS
WHERE           NOM = 'TOTO'
```

Exemple 2:

- **Supprimer tous les étudiants ayant la même moyenne que l'étudiant ayant pour nom : TOTO.**

```
DELETE FROM     ETUDIANT
WHERE           MOYENNE IN
                ( SELECT MOYENNE
                  FROM   ETUDIANTS
                  WHERE  NOM = 'TOTO')
```

- **Une commande DELETE sans clause WHERE implique la suppression de toutes les lignes de la table.**

Utilisation des vues (VIEWS)

- Les vues sont en quelque sorte des fenêtres à travers lesquelles on peut voir les données de la base.
- Les vues ne contiennent pas de données mais c'est tout comme et peuvent être manipulées comme si c'étaient des tables.
- Du fait que les vues n'occupent pas d'espace mémoire, elles sont appelées **TABLES VIRTUELLES**.
- Une vue **est dérivée d'une ou plusieurs tables réelles**.
- Du fait que le résultat d'une requête est lui-même une table, une vue peut donc être définie grâce à une requête.

Création d'une vue : la commande CREATE VIEW

- Le format général d'une requête de création d'une vue est :

```
CREATE VIEW  <nom de la vue>
AS           <requête de création>
```

- **On peut utiliser toute la puissance de SQL pour définir une VUE.**

Exemple :

- On veut créer une vue MOYENNE_ETUDIANTS à partir de la table ETUDIANTS qui contiendra uniquement les noms et les moyennes des étudiants dont le nom commence par 'B' et la moyenne inférieure à 10.

```
CREATE VIEW  MOYENNE_ETUDIANTS
AS
        SELECT      NOM,MOYENNE
        FROM        ETUDIANTS
        WHERE       NOM LIKE 'B%'
        AND         MOYENNE < 10.0
```


Exemple 2 :

- On veut créer une vue RECHERCHE à partir des tables EMPLOYES , DEPARTEMENTS contenant uniquement les noms , fonctions et salaires des employés du département 'RECHERCHES' dont le salaire est supérieur à 1000 ?

```
CREATE VIEW RECHERCHE
AS
    SELECT   Nom_Emp , Fonction , Salaire
    FROM     EMPLOYES , DEPARTEMENTS
    WHERE    EMPLOYES•Num_Dept= DEPARTEMENTS•Num_Dept
    AND      Nom_Dept = 'RECHERCHE'
    AND      Salaire > 1000
```

Interrogation d'une vue

- On peut utiliser dans toute requête SQL le nom d'une vue,
- on peut aussi utiliser toute la puissance du langage SQL **pour interroger une vue.**

Exemple avec la vue MOYENNE_ETUDIANTS :

- Quels sont les nom et les moyennes de tous les étudiants dont le nom se termine par 'R'

```
SELECT   NOM, MOYENNE
FROM     MOYENNE_ETUDIANTS
WHERE    NOM LIKE '%R'
```

- Il faut souligner que les changements qui sont faits dans les tables sont directement visibles dans les vues qui ont été définies à partir de ces tables.

- On peut ajouter, mettre à jour ou supprimer des lignes dans une vue de la même façon qu'avec une table.

☝ Cependant certaines opérations (comme l'insertion) sur une vue engendrent des problèmes au niveau de la répercussion de ces opérations sur les tables qui ont servi à définir cette vue.

❖ **C'est le problème de propagation des mises à jour sur une vue.**

Exemple :

- Insérer l'étudiant de NOM = 'BACHIR' et de MOYENNE = 12 dans la vue MOYENNE_ETUDIANTS

```
INSERT INTO     MOYENNE_ETUDIANTS
VALUES          ('BACHIR',12)
```

- Or la table ETUDIANTS qui a servi à la création de cette vue a le schéma suivant :

ETUDIANTS(NUMERO, NOM, MOYENNE, AGE)

- Donc l'insertion du tuple ('BACHIR', 12) dans la vue ne peut se répercuter dans la table ETUDIANTS que sous la forme du tuple suivant : (**NULL**,'BACHIR',12,**NULL**)
- ou NULL correspond à une valeur indéfinie.
- Ceci pose un problème car l'attribut NUMERO a été défini comme NOT NULL lors de la création de la table ETUDIANTS

⇒ **on ne pourra donc pas insérer** le tuple (NULL,'BACHIR',12,NULL) dans la table.

⇒ Donc on ne peut pas répercuter l'insertion faite sur la vue au niveau de la table qui a permis de créer cette vue.

❖ **Le problème se complique encore plus si la vue a été obtenue par jointure entre plusieurs tables.**

Conclusion

- Le langage SQL est a l'heure actuelle le langage le plus utilisé dans le domaine des bases de données.
- Il constitue le support idéal au développement des L4G (langages de 4ieme génération).
- Un L4G est un outil de productivité permettant de développer des applications complètes sans programmation. Il est construit a partir d'un langage relationnel comme SQL.
- Certains SGBD comme ORACLE offrent des outils dans ce sens et qui sont tous développés autour de SQL. Ceci inclue :

  - un générateurs d'états
  - un tableur
  - un utilitaire de sortie graphique
  - un générateur d'application
  - etc.

- L'utilisation de SQL et des utilitaires associés ouvre le développement d'applications a de nouveaux utilisateurs (non spécialistes) tout en améliorant les performances (temps de développement, de maintenance et de flexibilité).
- Avec un véritable L4G, le développeur d'application n'est plus un programmeur au sens classique du terme (i.e. au sens des L3G procéduraux).
- Rien ne remplacerais l'apprentissage du langage par la pratique en utilisant n'importe quel SGBD relationnel dont on peut disposer.

  - Le SGBD Microsoft ACCESS
  - Le SGBD MySql. gratuitement et fonctionnant dans une architecture Client/Serveur